

---

# Chapter 1. API Variation

## Table of Contents

1. Liste des schémas .....	4
2. admin .....	5
2.1. Fonctions .....	5
3. cron .....	31
3.1. Fonctions .....	31
4. document .....	32
4.1. Description .....	32
4.2. Tables .....	32
4.2.1. document.document .....	32
4.2.2. document.document_secteur .....	33
4.2.3. document.document_type .....	33
4.2.4. document.document_type_etablissement .....	33
4.2.5. document.document_type_secteur .....	33
4.2.6. document.document_usager .....	34
4.2.7. document.documents .....	34
4.2.8. document.documents_secteur .....	34
4.3. Types .....	34
4.3.1. document.document_document_secteur_liste_details .....	34
4.3.2. document.document_documents_groupe_liste .....	34
4.3.3. document.document_documents_secteur_liste_details .....	34
4.3.4. document.document_documents_liste_details .....	35
4.4. Fonctions .....	35
5. events .....	67
5.1. Description .....	67
5.2. Tables .....	67
5.2.1. events.agressources .....	67
5.2.2. events.agressources_secteur .....	67
5.2.3. events.categorie_events .....	68
5.2.4. events.event .....	68
5.2.5. events.event_memo .....	69
5.2.6. events.event_personne .....	69
5.2.7. events.event_ressource .....	69
5.2.8. events.event_type .....	69
5.2.9. events.event_type_etablissement .....	70
5.2.10. events.event_type_secteur .....	70
5.2.11. events.events .....	70
5.2.12. events.events_categorie .....	71
5.2.13. events.secteur_event .....	71
5.2.14. events.secteur_events .....	71
5.3. Types .....	71
5.3.1. events.events_document_liste .....	71
5.3.2. events.events_echeance_liste .....	71
5.3.3. events.events_event_liste .....	71
5.3.4. events.events_event_type_list_all .....	71
5.3.5. events.events_event_type_list_par_evs .....	71
5.3.6. events.events_groupe_liste .....	72
5.3.7. events.events_groupe_liste2 .....	72
5.3.8. events.events_agressources_list_details .....	72
5.3.9. events.events_event_avec_ressource_liste .....	72
5.3.10. events.events_event_bilan .....	72
5.3.11. events.events_events_liste_details .....	72

---

5.4. Fonctions .....	72
6. liste .....	135
6.1. Description .....	135
6.2. Tables .....	135
6.2.1. liste.champ .....	135
6.2.2. liste.default .....	136
6.2.3. liste.liste .....	137
6.2.4. liste.supp .....	137
6.3. Types .....	137
6.3.1. liste.liste_champ_liste_details .....	137
6.4. Fonctions .....	137
7. localise .....	166
7.1. Description .....	166
7.2. Tables .....	166
7.2.1. localise.localisation_secteur .....	166
7.2.2. localise.terme .....	166
7.3. Types .....	167
7.3.1. localise.localise_terme_liste_details .....	167
7.4. Fonctions .....	167
8. lock .....	174
8.1. Description .....	174
8.2. Tables .....	174
8.2.1. lock.fiche .....	174
8.3. Fonctions .....	174
9. login .....	180
9.1. Description .....	180
9.2. Tables .....	181
9.2.1. login.grouputil .....	181
9.2.2. login.grouputil_groupe .....	181
9.2.3. login.grouputil_portail .....	181
9.2.4. login.token .....	181
9.2.5. login.utilisateur .....	182
9.2.6. login.utilisateur_grouputil .....	183
9.3. Types .....	183
9.3.1. login.utilisateur_login2 .....	183
9.3.2. login.utilisateur_liste_details_configuration .....	183
9.3.3. login.utilisateur_usagers_liste .....	183
9.4. Fonctions .....	183
10. meta .....	213
10.1. Description .....	213
10.2. Tables .....	213
10.2.1. meta.categorie .....	213
10.2.2. meta.dirinfo .....	214
10.2.3. meta.entite .....	214
10.2.4. meta.groupe_infos .....	214
10.2.5. meta.info .....	215
10.2.6. meta.info_aide .....	216
10.2.7. meta.info_groupe .....	216
10.2.8. meta.infos_type .....	216
10.2.9. meta.lien_familial .....	217
10.2.10. meta.menu .....	217
10.2.11. meta.metier .....	217
10.2.12. meta.metier_entite .....	217
10.2.13. meta.metier_secteur .....	218
10.2.14. meta.portail .....	218
10.2.15. meta.secteur .....	218
10.2.16. meta.secteur_type .....	219
10.2.17. meta.selection .....	219

---

10.2.18. meta.selection_entree .....	219
10.2.19. meta.sousmenu .....	220
10.2.20. meta.topmenu .....	220
10.2.21. meta.topsousmenu .....	220
10.3. Types .....	221
10.3.1. meta.meta_info_groupe_liste .....	221
10.3.2. meta.meta_sousmenus_liste_depuis_topmenu .....	221
10.3.3. meta.metier_liste_details .....	221
10.3.4. meta.meta_info_usage .....	221
10.4. Fonctions .....	221
11. notes .....	352
11.1. Description .....	352
11.2. Tables .....	352
11.2.1. notes.note .....	352
11.2.2. notes.note_destinataire .....	353
11.2.3. notes.note_groupe .....	353
11.2.4. notes.note_theme .....	353
11.2.5. notes.note_usager .....	354
11.2.6. notes.notes .....	354
11.2.7. notes.theme .....	354
11.2.8. notes.theme_portail .....	354
11.3. Types .....	354
11.3.1. notes.note_destinataire_derniers_par_utilisateur .....	354
11.3.2. notes.note_destinataires_liste .....	355
11.3.3. notes.note_usagers_liste .....	355
11.3.4. notes.notes_note_boite_envoi_liste .....	355
11.3.5. notes.notes_note_boite_reception_liste .....	355
11.3.6. notes.notes_note_mesnotes .....	355
11.3.7. notes.notes_theme_liste_details .....	355
11.4. Fonctions .....	355
12. permission .....	378
12.1. Description .....	378
12.2. Tables .....	378
12.2.1. permission.droit_ajout_entite_portail .....	378
12.3. Fonctions .....	379
13. procedure .....	381
13.1. Description .....	381
13.2. Tables .....	381
13.2.1. procedure.procedure .....	381
13.2.2. procedure.procedure_affectation .....	381
13.3. Types .....	382
13.3.1. procedure.procedure_procedure_details .....	382
13.4. Fonctions .....	382
14. public .....	391
14.1. Description .....	391
14.2. Tables .....	391
14.2.1. public.adresse .....	391
14.2.2. public.etablissement .....	392
14.2.3. public.etablissement_secteur .....	392
14.2.4. public.etablissement_secteur_edit .....	393
14.2.5. public.groupe .....	393
14.2.6. public.groupe_info_secteur .....	396
14.2.7. public.groupe_secteur .....	396
14.2.8. public.personne .....	396
14.2.9. public.personne_etablissement .....	397
14.2.10. public.personne_groupe .....	397
14.2.11. public.personne_info .....	398
14.2.12. public.personne_info_boolean .....	398

14.2.13. public.personne_info_date .....	399
14.2.14. public.personne_info_integer .....	399
14.2.15. public.personne_info_integer2 .....	400
14.2.16. public.personne_info_lien_familial .....	400
14.2.17. public.personne_info_text .....	401
14.2.18. public.personne_info_varchar .....	401
14.2.19. public.pgprocedures_stats .....	401
14.2.20. public.secteur_infos .....	401
14.3. Types .....	402
14.3.1. public.contact_recherche .....	402
14.3.2. public.etablissement_liste_details .....	402
14.3.3. public.famille_recherche .....	402
14.3.4. public.groupe_liste .....	402
14.3.5. public.groupe_liste_details .....	402
14.3.6. public.integer2 .....	402
14.3.7. public.personne_cherche .....	402
14.3.8. public.personne_contact_liste .....	402
14.3.9. public.personne_info_boolean_histo .....	402
14.3.10. public.personne_info_contact_histo .....	402
14.3.11. public.personne_info_date_histo .....	402
14.3.12. public.personne_info_integer_get_multiple_details .....	402
14.3.13. public.personne_info_integer_histo .....	402
14.3.14. public.personne_info_varchar_histo .....	402
14.3.15. public.pgprocedures_search_arguments .....	402
14.3.16. public.pgprocedures_search_function .....	402
14.3.17. public.prise_en_charge_select .....	402
14.3.18. public.groupe_cherche .....	403
14.4. Fonctions .....	403
15. ressource .....	568
15.1. Description .....	568
15.2. Tables .....	568
15.2.1. ressource.ressource .....	568
15.2.2. ressource.ressource_secteur .....	568
15.3. Types .....	569
15.3.1. ressource.ressource_liste_details .....	569
15.4. Fonctions .....	569

## 1. Liste des schémas

admin

Section 2, “admin”

cron

Section 3, “cron”

document

Données des documents rattachés aux usagers.

Section 4, “document”

events

Données des événements rattachés aux usagers.

Section 5, “events”

liste

Configuration des pages liste.

Section 6, “liste”

localise

Système de localisation. A ce point, il est possible de localiser un terme pour un secteur particulier.

Section 7, “localise”

lock

Système de verrouillage de fiches.

Section 8, “lock”

login

Procédures d'authentification de l'utilisateur. Gestion des utilisateurs et groupes d'utilisateurs.

Section 9, “login”

meta

Informations de construction de l'interface. Informations de base.

Section 10, “meta”

notes

Données des notes écrites par les utilisateurs.

Section 11, “notes”

permission

Enregistrement des diverses permissions.

Section 12, “permission”

procedure

Documentations à afficher sur diverses pages.

Section 13, “procedure”

public

Données des établissements, groupes et personnes.

Section 14, “public”

ressource

Ressources à affecter à des événements.

Section 15, “ressource”

## **2. admin**

### **2.1. Fonctions**

## Name

admin\_categorie\_supprime\_tout

## Synopsis

```
int4 admin_categorie_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    cat integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM meta.categorie;
    FOR cat IN SELECT cat_id FROM meta.categorie LOOP
        PERFORM meta.meta_categorie_delete (prm_token, cat);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_document\_supprime\_tout

## Synopsis

```
int4 admin_document_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    doc integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM document.document;
    FOR doc IN SELECT doc_id FROM document.document LOOP
        PERFORM document.document_document_supprime (prm_token, doc);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_document\_type\_supprime\_tout

## Synopsis

```
int4 admin_document_type_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    dty integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM document.document_type;
    FOR dty IN SELECT dty_id FROM document.document_type LOOP
        PERFORM document.document_document_type_supprime (prm_token, dty);
    END LOOP;
    RETURN ret;
END;
```



## Name

admin\_documents\_supprime\_tout

## Synopsis

```
int4 admin_documents_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
  ret integer;
  dos integer;
BEGIN
  SELECT COUNT(*) INTO ret FROM document.documents;
  FOR dos IN SELECT dos_id FROM document.documents LOOP
    PERFORM document.document_documents_supprime (prm_token, dos);
  END LOOP;
  RETURN ret;
END;
```

## Name

admin\_etablissement\_supprime\_tout

## Synopsis

```
int4 admin_etablissement_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    eta integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM public.etablissement;
    FOR eta IN SELECT eta_id FROM public.etablissement LOOP
        PERFORM public.etablissement_supprime (prm_token, eta);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_event\_supprime\_tout

## Synopsis

```
int4 admin_event_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    eve integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM events.event;
    FOR eve IN SELECT eve_id FROM events.event LOOP
        PERFORM events.events_event_supprime (prm_token, eve);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_event\_type\_etablissement\_supprime\_tout

## Synopsis

```
int4 admin_event_type_etablissement_supprime_tout();
```

## Source

```
DECLARE
    ret integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM events.event_type_etablissement;
    DELETE FROM events.event_type_etablissement;
    RETURN ret;
END;
```

## Name

admin\_event\_type\_supprime\_tout

## Synopsis

```
int4 admin_event_type_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
  ret integer;
  ety integer;
BEGIN
  SELECT COUNT(*) INTO ret FROM events.event_type;
  FOR ety IN SELECT ety_id FROM events.event_type LOOP
    PERFORM events.events_event_type_supprime (prm_token, ety);
  END LOOP;
  RETURN ret;
END;
```

## Name

admin\_events\_supprime\_tout

## Synopsis

```
int4 admin_events_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    evs integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM events.events;
    FOR evs IN SELECT evs_id FROM events.events LOOP
        PERFORM events.events_events_supprime (prm_token, evs);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_fiche\_supprime\_tout

## Synopsis

```
int4 admin_fiche_supprime_tout();
```

## Source

```
DECLARE
    ret integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM lock.fiche;
    DELETE FROM lock.fiche;
    RETURN ret;
END;
```

## Name

admin\_groupe\_supprime\_tout

## Synopsis

```
int4 admin_groupe_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    grp integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM public.groupe;
    FOR grp IN SELECT grp_id FROM public.groupe LOOP
        PERFORM public.groupe_supprime (prm_token, grp);
    END LOOP;
    RETURN ret;
END;
```



## Name

admin\_grouputil\_supprime\_tout

## Synopsis

```
int4 admin_grouputil_supprime_tout();
```

## Source

```
DECLARE
    ret integer;
    gut integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM login.grouputil;
    FOR gut IN SELECT gut_id FROM login.grouputil LOOP
        PERFORM login.login_grouputil_supprime (gut);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_info\_supprime\_tout

## Synopsis

```
void admin_info_supprime_tout();
```

## Source

```
BEGIN
  DELETE FROM meta.info_aide;
  DELETE FROM meta.info;
  DELETE FROM meta.selection_entree;
  DELETE FROM meta.selection;
  DELETE FROM meta.dirinfo;
END;
```

## Name

admin\_liste\_supprime\_tout

## Synopsis

```
int4 admin_liste_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    lis integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM liste.liste;
    FOR lis IN SELECT lis_id FROM liste.liste LOOP
        PERFORM liste.liste_liste_supprime (prm_token, lis);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_metier\_supprime\_tout

## Synopsis

```
int4 admin_metier_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    met integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM meta.metier;
    FOR met IN SELECT met_id FROM meta.metier LOOP
        PERFORM meta.metier_supprime (prm_token, met);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_note\_supprime\_tout

## Synopsis

```
int4 admin_note_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    no integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM notes.note;
    FOR no IN SELECT not_id FROM notes.note LOOP
        PERFORM notes.notes_note_supprime (prm_token, no);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_notes\_supprime\_tout

## Synopsis

```
int4 admin_notes_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    nos integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM notes.notes;
    FOR nos IN SELECT nos_id FROM notes.notes LOOP
        PERFORM notes.notes_notes_supprime (prm_token, nos);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_personne\_supprime\_tout

## Synopsis

```
int4 admin_personne_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    per integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM personne;
    FOR per IN SELECT per_id FROM personne LOOP
        PERFORM personne_supprime (prm_token, per);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_portail\_supprime\_tout

## Synopsis

```
int4 admin_portail_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
  ret integer;
  por integer;
BEGIN
  SELECT COUNT(*) INTO ret FROM meta.portail;
  FOR por IN SELECT por_id FROM meta.portail LOOP
    PERFORM meta.meta_portail_delete_rec (prm_token, por);
  END LOOP;
  RETURN ret;
END;
```



## Name

admin\_procedure\_supprime\_tout

## Synopsis

```
int4 admin_procedure_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    pro integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM procedure.procedure;
    FOR pro IN SELECT pro_id FROM procedure.procedure LOOP
        PERFORM procedure.procedure_procedure_supprime (prm_token, pro);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_ressource\_supprime\_tout

## Synopsis

```
int4 admin_ressource_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
  ret integer;
  res integer;
BEGIN
  SELECT COUNT(*) INTO ret FROM ressource.ressource;
  FOR res IN SELECT res_id FROM ressource.ressource LOOP
    PERFORM ressource.ressource_supprime (prm_token, res);
  END LOOP;
  RETURN ret;
END;
```

## Name

admin\_secteur\_infos\_supprime\_tout

## Synopsis

```
void admin_secteur_infos_supprime_tout();
```

## Source

```
BEGIN
  DELETE FROM public.secteur_infos;
END;
```

## Name

admin\_supprime\_tout

## Synopsis

```
int4 admin_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
BEGIN
PERFORM admin.admin_document_supprime_tout(prm_token);
PERFORM admin.admin_event_supprime_tout(prm_token);
PERFORM admin.admin_fiche_supprime_tout();
PERFORM admin.admin_note_supprime_tout(prm_token);
PERFORM admin.admin_ressource_supprime_tout();
PERFORM admin.admin_utilisateur_supprime_tout(prm_token);
PERFORM admin.admin_grouputil_supprime_tout();
PERFORM admin.admin_personne_supprime_tout(prm_token);
PERFORM admin.admin_groupe_supprime_tout(prm_token);
PERFORM admin.admin_event_type_etablissement_supprime_tout();
PERFORM admin.admin_etablissement_supprime_tout(prm_token);
RETURN 1;
END;
```

## Name

admin\_terme\_supprime\_tout

## Synopsis

```
int4 admin_terme_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
  ret integer;
  ter integer;
BEGIN
  SELECT COUNT(*) INTO ret FROM localise.termes;
  FOR ter IN SELECT ter_id FROM localise.termes LOOP
    PERFORM localise.localise_terme_supprime (prm_token, ter);
  END LOOP;
  RETURN ret;
END;
```

## Name

admin\_theme\_supprime\_tout

## Synopsis

```
int4 admin_theme_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
    ret integer;
    the integer;
BEGIN
    SELECT COUNT(*) INTO ret FROM notes.theme;
    FOR the IN SELECT the_id FROM notes.theme LOOP
        PERFORM notes.notes_theme_supprime (prm_token, the);
    END LOOP;
    RETURN ret;
END;
```

## Name

admin\_utilisateur\_supprime\_tout — Supprime tous les utilisateurs.

## Synopsis

```
int4 admin_utilisateur_supprime_tout(prm_token);
```

```
int4 prm_token;
```

## Description

Supprime tous les utilisateurs. Entrée :

- *prm\_token* : Token d'authentification

Remarques : Nécessite les droits à la configuration "Établissement"

## Source

```
DECLARE
  ret integer;
  uti integer;
BEGIN
  SELECT COUNT(*) INTO ret FROM login.utilisateur;
  FOR uti IN SELECT uti_id FROM login.utilisateur LOOP
    PERFORM login.utilisateur_supprime (prm_token, uti);
  END LOOP;
  RETURN ret;
END;
```

## 3. cron

### 3.1. Fonctions

## Name

cron\_jour

## Synopsis

```
void cron_jour();
```

## Source

```
DECLARE
  row RECORD;
  tmp integer;
BEGIN
  FOR row IN
    SELECT per_id FROM personne WHERE ent_code = 'usager'
  LOOP
    SELECT * INTO tmp FROM meta.meta_statut_usager_calcule ('statut_usager', row.per_id);
  END LOOP;
END;
```

## 4. document

### 4.1. Description

Données des documents rattachés aux usagers. Il est possible de rattacher des documents à des usagers, et de les classer par secteur (thème) et par type de document. Il est possible de définir des types de documents et des les affecter à des secteurs (thèmes) particuliers. Chacun de ces types de documents pourront être utilisés ou non par chacun des établissements du réseau.

### 4.2. Tables

#### 4.2.1. document.document

Les documents

doc\_id (int4)

per\_id\_responsable (int4)

Clé étrangère vers personne.per\_id

dty\_id (int4)

Clé étrangère vers document\_type.dty\_id

doc\_titre (varchar)

doc\_statut (int4)

doc\_date\_obtention (date)

doc\_date\_realisation (date)

doc\_date\_validite (date)

doc\_description (text)

doc\_fichier (varchar)

doc\_date\_creation (timestampz)



uti\_id\_creation (int4)

Clé étrangère vers utilisateur.uti\_id

### **Liens vers cette table**

- document\_secteur.doc\_id
- document\_usager.doc\_id

#### **4.2.2. document.document\_secteur**

Rattachement d'un document à des secteurs

dse\_id (int4)

doc\_id (int4)

Clé étrangère vers document.doc\_id

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

#### **4.2.3. document.document\_type**

Types de documents

dtv\_id (int4)

dtv\_nom (varchar)

### **Liens vers cette table**

- document.dtv\_id
- document\_type\_etablissement.dtv\_id
- document\_type\_secteur.dtv\_id
- documents.dtv\_id

#### **4.2.4. document.document\_type\_etablissement**

Utilisation d'un type de document par un établissement

dte\_id (int4)

dtv\_id (int4)

Clé étrangère vers document\_type.dtv\_id

eta\_id (int4)

Clé étrangère vers etablissement.eta\_id

#### **4.2.5. document.document\_type\_secteur**

Affectation d'un type de document à des secteurs

dts\_id (int4)

dtv\_id (int4)  
Clé étrangère vers document\_type.dtv\_id

sec\_id (int4)  
Clé étrangère vers secteur.sec\_id

#### **4.2.6. document.document\_usager**

Rattachement d'un document à des usagers

dus\_id (int4)

doc\_id (int4)  
Clé étrangère vers document.doc\_id

per\_id\_usager (int4)  
Clé étrangère vers personne.per\_id

#### **4.2.7. document.documents**

Configuration des pages de documents disponibles pour placer dans le menu principal ou usager

dos\_id (int4)

dos\_titre (varchar)

dtv\_id (int4)  
Clé étrangère vers document\_type.dtv\_id

#### **Liens vers cette table**

- documents\_secteur.dos\_id

#### **4.2.8. document.documents\_secteur**

Spécialisation des pages de documents à certains secteurs

dss\_id (int4)

dos\_id (int4)  
Clé étrangère vers documents.dos\_id

sec\_id (int4)  
Clé étrangère vers secteur.sec\_id

### **4.3. Types**

#### **4.3.1. document.document\_document\_secteur\_liste\_details**

#### **4.3.2. document.document\_documents\_groupe\_liste**

#### **4.3.3. document.document\_documents\_secteur\_liste\_details**

#### **4.3.4. document.document\_documents\_liste\_details**

### **4.4. Fonctions**

## Name

document\_document\_get — Retourne les informations concernant un document.

## Synopsis

```
document document_document_get(prm_token, prm_doc_id);
```

```
int4 prm_token;
```

```
int4 prm_doc_id;
```

## Description

Retourne les informations concernant un document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_doc\_id* : Identifiant du document

## Source

```
DECLARE
    ret document.document;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM document.document WHERE doc_id = prm_doc_id;
    RETURN ret;
END;
```

## Name

document\_document\_liste — Retourne une liste de documents, filtrée selon plusieurs paramètres.

## Synopsis

```
setof document document_document_liste(prm_token, prm_dos_id, prm_per_id,
prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_dos_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne une liste de documents, filtrée selon plusieurs paramètres. Entrées :

- prm\_token : Token d'authentification
- prm\_dos\_id : Identifiant de la configuration d'une page de documents, pour utiliser les filtres de cette page
- prm\_per\_id : Retourne uniquement les documents rattachés à cet usager
- prm\_start : Date de début de recherche
- prm\_end : Date de fin de recherche
- prm\_grp\_id : Retourne uniquement les documents rattachés aux usagers de ce groupe
- prm\_per\_ids : Retourne uniquement les documents rattachés à ces usagers

## Source

```
DECLARE
  row document.document;
  p_start timestamp;
  p_end timestamp;

BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  if prm_start NOTNULL THEN
    p_start = prm_start;
  ELSE
    p_start = timestamp '-INFINITY';
  END IF;
  if prm_end NOTNULL THEN
    p_end = prm_end;
  ELSE
    p_end = timestamp 'INFINITY';
  END IF;

  FOR row IN
    SELECT DISTINCT document.*
      FROM document.document
      INNER JOIN document.document_secteur USING(doc_id)
      INNER JOIN document.documents_secteur USING(sec_id)
      INNER JOIN document.documents USING(dos_id)
      INNER JOIN document.document_usager USING(doc_id)
      LEFT JOIN personne_groupe on personne_groupe.per_id = document_usager.per_id_usager
     WHERE dos_id = prm_dos_id
      AND (documents.dty_id ISNULL OR documents.dty_id = document.dty_id)
```

```
AND (prm_per_id ISNULL OR prm_per_id = per_id_usager)
AND ((prm_start ISNULL AND prm_end ISNULL) OR doc_date_obtention BETWEEN p_start AND p_end OR doc_d
AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)
AND (prm_per_ids ISNULL OR document_usager.per_id_usager = ANY (prm_per_ids))
LOOP
RETURN NEXT row;
END LOOP;
END;
```

## Name

document\_document\_save — Crée un nouveau document ou modifie les informations d'un document existant.

## Synopsis

```
int4 document_document_save(prm_token, prm_doc_id, prm_per_id_responsable,
prm_dty_id, prm_titre, prm_statut, prm_date_obtention, prm_date_realisation,
prm_date_validite, prm_description);
```

```
int4 prm_token;
int4 prm_doc_id;
int4 prm_per_id_responsable;
int4 prm_dty_id;
varchar prm_titre;
int4 prm_statut;
date prm_date_obtention;
date prm_date_realisation;
date prm_date_validite;
text prm_description;
```

## Description

Crée un nouveau document ou modifie les informations d'un document existant. Entrées :

- *prm\_token* : Token d'authentification, le créateur du document sera l'utilisateur authentifié par ce token lors de la création d'un document
- *prm\_doc\_id* : Identifiant du document à modifier, ou NULL pour créer un document
- *prm\_per\_id\_responsable* : Identifiant du personnel responsable
- *prm\_dty\_id* : Identifiant du type de document
- *prm\_titre* : Intitulé du document
- *prm\_statut* : Statut du document : 1=À faire, 2=Demandé, 3=Existant
- *prm\_date\_obtention* : Date d'obtention du document
- *prm\_date\_realisation* : Date de réalisation du document
- *prm\_date\_validite* : Date de validité du document
- *prm\_description* : Description du document

## Source

```
DECLARE
    uti integer;
    ret integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    IF prm_doc_id NOTNULL THEN
        -- TODO modif uti_id_modif ...
        UPDATE document.document SET
            per_id_responsable = prm_per_id_responsable,
            dty_id = prm_dty_id,
            doc_titre = prm_titre,
            doc_statut = prm_statut,
            doc_date_obtention = prm_date_obtention,
            doc_date_realisation = prm_date_realisation,
```

```
    doc_date_validite = prm_date_validite,  
    doc_description = prm_description  
    WHERE doc_id = prm_doc_id;  
    RETURN prm_doc_id;  
ELSE  
    INSERT INTO document.document (per_id_responsable, dty_id, doc_titre, doc_statut, doc_date_obtention,  
        VALUES (prm_per_id_responsable, prm_dty_id, prm_titre, prm_statut, prm_date_obtention, prm_date_rea  
        RETURNING doc_id INTO ret;  
    RETURN ret;  
END IF;  
END;
```



## Name

`document_document_secteur_liste_details` — Retourne la liste des secteurs (thèmes) auxquels un document est rattaché.

## Synopsis

```
setof                                document_document_secteur_liste_details
document_document_secteur_liste_details(prm_token, prm_doc_id);
```

```
int4 prm_token;
int4 prm_doc_id;
```

## Description

Retourne la liste des secteurs (thèmes) auxquels un document est rattaché. Entrées :

- `prm_token` : Token d'authentification
- `prm_doc_id` : Identifiant du document

Retour :

- `dse_id` : Identifiant du rattachement du document au secteur
- `sec_id` : Identifiant du secteur
- `sec_nom` : nom du secteur

## Source

```
DECLARE
  row document.document_document_secteur_liste_details;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT dse_id, sec_id, sec_nom
    FROM document.document_secteur
    INNER JOIN meta.secteur USING(sec_id)
    WHERE doc_id = prm_doc_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

document\_document\_secteur\_save — Rattache un document à une liste de secteurs (thèmes).

## Synopsis

```
void document_document_secteur_save(prm_token, prm_doc_id, prm_sec_ids);
```

```
int4 prm_token;
```

```
int4 prm_doc_id;
```

```
int4[] prm_sec_ids;
```

## Description

Rattache un document à une liste de secteurs (thèmes). Entrées :

- *prm\_token* : Token d'authentification
- *prm\_doc\_id* : Identifiant du document
- *prm\_sec\_ids* : Tableau d'identifiants de secteurs

## Source

```
DECLARE
    row document.document_secteur;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM document.document_secteur WHERE doc_id = prm_doc_id
    LOOP
        IF NOT row.sec_id = ANY (prm_sec_ids) THEN
            DELETE FROM document.document_secteur WHERE dse_id = row.dse_id;
        END IF;
    END LOOP;
    FOR i IN 1 .. array_upper (prm_sec_ids, 1) LOOP
        IF NOT EXISTS (SELECT 1 FROM document.document_secteur WHERE doc_id = prm_doc_id AND sec_id = prm_sec
            INSERT INTO document.document_secteur (doc_id, sec_id) VALUES (prm_doc_id, prm_sec_ids[i]);
        END IF;
    END LOOP;
END;
```

## Name

document\_document\_set\_fichier — Rattache un fichier à un document.

## Synopsis

```
void document_document_set_fichier(prm_token, prm_doc_id, prm_fichier);
```

```
int4 prm_token;
```

```
int4 prm_doc_id;
```

```
varchar prm_fichier;
```

## Description

Rattache un fichier à un document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_doc\_id* : Identifiant du document
- *prm\_fichier* : Nom du fichier à rattacher au document

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  UPDATE document.document SET doc_fichier = prm_fichier WHERE doc_id = prm_doc_id;
END;
```

## Name

document\_document\_supprime — Supprime un document.

## Synopsis

```
void document_document_supprime(prm_token, prm_doc_id);
```

```
int4 prm_token;
```

```
int4 prm_doc_id;
```

## Description

Supprime un document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_doc\_id* : Identifiant du document à supprimer

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  DELETE FROM document.document_secteur WHERE doc_id = prm_doc_id;
  DELETE FROM document.document_usager WHERE doc_id = prm_doc_id;
  DELETE FROM document.document WHERE doc_id = prm_doc_id;
END;
```

## Name

document\_document\_type\_ajoute — Ajoute un nouveau type de document.

## Synopsis

```
int4 document_document_type_ajoute(prm_token, prm_nom);
```

```
int4 prm_token;  
varchar prm_nom;
```

## Description

Ajoute un nouveau type de document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_nom* : Nom du nouveau type de document

Retour :

- Identifiant du type de document créé

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, TRUE);  
    INSERT INTO document.document_type (dtm_nom) VALUES (prm_nom) RETURNING dtm_id INTO ret;  
    RETURN ret;  
END;
```

## Name

`document_document_type_etablissement_get` — Retourne TRUE si un type de document est affecté à un établissement, FALSE sinon.

## Synopsis

```
bool    document_document_type_etablissement_get(prm_token,    prm_dty_id,  
prm_eta_id);
```

```
int4    prm_token;  
int4    prm_dty_id;  
int4    prm_eta_id;
```

## Description

Retourne TRUE si un type de document est affecté à un établissement, FALSE sinon. Entrées :

- `prm_token` : Token d'authentification
- `prm_dty_id` : Identifiant du type de document
- `prm_eta_id` : Identifiant de l'établissement

Remarques : Nécessite les droits à la configuration "Établissement".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  RETURN EXISTS (SELECT 1 FROM document.document_type_etablissement WHERE dty_id = prm_dty_id AND eta_id  
END;
```

## Name

document\_document\_type\_etablissement\_set — Indique si un type de document est affecté à un établissement.

## Synopsis

```
void    document_document_type_etablissement_set(prm_token,    prm_dty_id,  
prm_eta_id, prm_b);
```

```
int4    prm_token;  
int4    prm_dty_id;  
int4    prm_eta_id;  
bool    prm_b;
```

## Description

Indique si un type de document est affecté à un établissement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dty\_id* : Identifiant du type de document
- *prm\_eta\_id* : Identifiant de l'établissement
- *prm\_b* : TRUE si le type de document est affecté à l'établissement, FALSE sinon

Remarques : Nécessite les droits à la configuration "Établissement".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  IF EXISTS (SELECT 1 FROM document.document_type_etablissement WHERE dty_id = prm_dty_id AND eta_id = prm_eta_id)  
    DELETE FROM document.document_type_etablissement WHERE dty_id = prm_dty_id AND eta_id = prm_eta_id;  
  ELSIF NOT EXISTS (SELECT 1 FROM document.document_type_etablissement WHERE dty_id = prm_dty_id AND eta_id = prm_eta_id)  
    INSERT INTO document.document_type_etablissement (dty_id, eta_id) VALUES (prm_dty_id, prm_eta_id);  
  END IF;  
END;
```

## Name

document\_document\_type\_get — Retourne les informations d'un type de document.

## Synopsis

```
document_type document_document_type_get(prm_token, prm_dty_id);
```

```
int4 prm_token;  
int4 prm_dty_id;
```

## Description

Retourne les informations d'un type de document. Entrées :

- prm\_token : Token d'authentification
- prm\_dty\_id : Identifiant du type de document

## Source

```
DECLARE  
    ret document.document_type;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM document.document_type WHERE dty_id = prm_dty_id;  
    RETURN ret;  
END;
```



## Name

`document_document_type_liste` — Retourne la liste des types de documents applicables à un document, étant donné les secteurs auxquels est rattaché le document et un établissement.

## Synopsis

```
setof document_type document_document_type_liste(prm_token, prm_doc_id,
prm_eta_id);
```

```
int4 prm_token;
int4 prm_doc_id;
int4 prm_eta_id;
```

## Description

Retourne la liste des types de documents applicables à un document, étant donné les secteurs auxquels est rattaché le document et un établissement. Entrées :

- `prm_token` : Token d'authentification
- `prm_doc_id` : Identifiant du document
- `prm_eta_id` : Identifiant de l'établissement (les types de documents étant affectés ou non à un établissement)

## Source

```
DECLARE
    row document.document_type;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT document_type.*
        FROM document.document_type
        INNER JOIN document.document_type_secteur USING(dty_id)
        INNER JOIN document.document_secteur USING(sec_id)
        INNER JOIN document.document_type_etablissement USING(dty_id)
        WHERE doc_id = prm_doc_id AND (prm_eta_id ISNULL OR eta_id = prm_eta_id)
        group by document_type.dty_id
        having array_agg(sec_id) = (select array_agg(DISTINCT sec_id) FROM document.document_secteur where
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`document_document_type_liste_par_sec_ids` — Retourne la liste des types de documents rattachés à certains secteurs, et affectés à un établissement.

## Synopsis

```
setof document_type document_document_type_liste_par_sec_ids(prm_token,
prm_sec_ids, prm_eta_id);
```

```
int4 prm_token;
int4[] prm_sec_ids;
int4 prm_eta_id;
```

## Description

Retourne la liste des types de documents rattachés à certains secteurs, et affectés à un établissement. Entrées :

- `prm_token` : Token d'authentification
- `prm_sec_ids` : Tableau d'identifiant de secteurs (les types de documents listés seront ceux affectés à TOUS ces secteurs) OU NULL pour retourner tous les types de documents
- `prm_eta_id` : Identifiant de l'établissement auquel sont rattachés les types de documents, ou NULL pour retourner tous les types de documents

## Source

```
DECLARE
    row document.document_type;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    IF prm_sec_ids ISNULL THEN
        FOR row IN
            SELECT DISTINCT document_type.*
            FROM document.document_type
            LEFT JOIN document.document_type_etablissement USING(dty_id)
            WHERE (prm_eta_id ISNULL OR eta_id = prm_eta_id)
            ORDER BY dty_nom
        LOOP
            RETURN NEXT row;
        END LOOP;
    ELSE
        FOR row IN
            SELECT DISTINCT document_type.*
            FROM document.document_type
            INNER JOIN document.document_type_secteur USING(dty_id)
            LEFT JOIN document.document_type_etablissement USING(dty_id)
            WHERE (prm_eta_id ISNULL OR eta_id = prm_eta_id)
            GROUP BY document_type.dty_id
            HAVING array_agg(sec_id) @> prm_sec_ids
            ORDER BY dty_nom
        LOOP
            RETURN NEXT row;
        END LOOP;
    END IF;
END;
```

## Name

document\_document\_type\_secteur\_ajoute — Affecte un type de document à un secteur.

## Synopsis

```
int4      document_document_type_secteur_ajoute(prm_token,      prm_dty_id,  
prm_sec_id);
```

```
int4 prm_token;  
int4 prm_dty_id;  
int4 prm_sec_id;
```

## Description

Affecte un type de document à un secteur. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dty\_id* : Identifiant du type de document
- *prm\_sec\_id* : Identifiant du secteur

Remarques : Nécessite les droits à la configuration "Réseau".

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  INSERT INTO document.document_type_secteur (dty_id, sec_id) VALUES (prm_dty_id, prm_sec_id) RETURNING d  
  RETURN ret;  
END;
```

## Name

document\_document\_type\_secteur\_list — Retourne la liste des secteurs auxquels est affecté un type de document.

## Synopsis

```
setof secteur document_document_type_secteur_list(prm_token, prm_dty_id);
```

```
int4 prm_token;
```

```
int4 prm_dty_id;
```

## Description

Retourne la liste des secteurs auxquels est affecté un type de document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dty\_id* : Identifiant du type de document

Remarques : Nécessite les droits à la configuration "Établissement" ou "Réseau".

## Source

```
DECLARE
    row meta.secteur;
BEGIN
    PERFORM login._token_assert (prm_token, TRUE, TRUE);
    FOR row IN
        SELECT secteur.* FROM meta.secteur
            INNER JOIN document.document_type_secteur USING(sec_id)
            WHERE dty_id = prm_dty_id
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

document\_document\_type\_secteur\_supprime — Supprime l'affectation un type de document à un secteur.

## Synopsis

```
void      document_document_type_secteur_supprime(prm_token,      prm_dty_id,  
prm_sec_id);
```

```
int4 prm_token;  
int4 prm_dty_id;  
int4 prm_sec_id;
```

## Description

Supprime l'affectation un type de document à un secteur. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dty\_id* : Identifiant du type de document
- *prm\_sec\_id* : Identifiant du secteur

Remarques : Nécessite les droits à la configuration "Réseau".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  DELETE FROM document.document_type_secteur WHERE dty_id = prm_dty_id AND sec_id = prm_sec_id;  
END;
```

## Name

document\_document\_type\_set\_nom — Modifie le nom d'un type de document.

## Synopsis

```
void document_document_type_set_nom(prm_token, prm_dty_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_dty_id;  
varchar prm_nom;
```

## Description

Modifie le nom d'un type de document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dty\_id* : Identifiant du type de document
- *prm\_nom* : Nouveau nom du type de document

Remarques : Nécessite les droits à la configuration "Réseau".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE document.document_type SET dty_nom = prm_nom WHERE dty_id = prm_dty_id;  
END;
```

## Name

document\_document\_type\_supprime — Supprime un type de document.

## Synopsis

```
void document_document_type_supprime(prm_token, prm_dty_id);
```

```
int4 prm_token;
```

```
int4 prm_dty_id;
```

## Description

Supprime un type de document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dty\_id* : Identifiant du type de document à supprimer.

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  DELETE FROM document.document_type_secteur WHERE dty_id = prm_dty_id;
  DELETE FROM document.document_type WHERE dty_id = prm_dty_id;
END;
```

## Name

document\_document\_usager\_liste — Retourne la liste des usagers auxquels est rattaché un document.

## Synopsis

```
setof document_usager document_document_usager_liste(prm_token, prm_doc_id);
```

```
int4 prm_token;
```

```
int4 prm_doc_id;
```

## Description

Retourne la liste des usagers auxquels est rattaché un document. Entrées :

- prm\_token : Token d'authentification
- prm\_doc\_id : Identifiant du document

## Source

```
DECLARE
    row document.document_usager;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM document.document_usager WHERE doc_id = prm_doc_id
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```



## Name

document\_document\_usager\_save — Modifie la liste des usagers auxquels est rattaché un document.

## Synopsis

```
void document_document_usager_save(prm_token, prm_doc_id, prm_per_ids);
```

```
int4 prm_token;
```

```
int4 prm_doc_id;
```

```
int4[] prm_per_ids;
```

## Description

Modifie la liste des usagers auxquels est rattaché un document. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_doc\_id* : Identifiant du document
- *prm\_per\_ids* : Tableau d'identifiants d'usagers

## Source

```
DECLARE
    row document.document_usager;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM document.document_usager WHERE doc_id = prm_doc_id
    LOOP
        IF NOT row.per_id_usager = ANY (prm_per_ids) THEN
            DELETE FROM document.document_usager WHERE dus_id = row.dus_id;
        END IF;
    END LOOP;
    FOR i IN 1 .. array_upper (prm_per_ids, 1) LOOP
        IF NOT EXISTS (SELECT 1 FROM document.document_usager WHERE doc_id = prm_doc_id AND per_id_usager = p
            INSERT INTO document.document_usager (doc_id, per_id_usager) VALUES (prm_doc_id, prm_per_ids[i]);
        END IF;
    END LOOP;
END;
```

## Name

`document_documents_get` — Retourne les informations de configuration d'une page de documents.

## Synopsis

```
documents document_documents_get(prm_token, prm_dos_id);
```

```
int4 prm_token;
```

```
int4 prm_dos_id;
```

## Description

Retourne les informations de configuration d'une page de documents. Entrées :

- `prm_token` : Token d'authentification
- `prm_dos_id` : Identifiant de la configuration de page de documents

## Source

```
DECLARE
    ret document.documents;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM document.documents WHERE dos_id = prm_dos_id;
    RETURN ret;
END;
```

## Name

`document_documents_groupe_liste` — Retourne la liste des groupes en relation avec une page de documents (en considérant les secteurs du groupe et les secteurs de la configuration de la page).

## Synopsis

```
setof                               document_documents_groupe_liste
document_documents_groupe_liste(prm_token, prm_dos_id);
```

```
int4 prm_token;
int4 prm_dos_id;
```

## Description

Retourne la liste des groupes en relation avec une page de documents (en considérant les secteurs du groupe et les secteurs de la configuration de la page). Entrées :

- `prm_token` : Token d'authentification
- `prm_dos_id` : Identifiant de la configuration de page de documents.

## Source

```
DECLARE
  row document.document_documents_groupe_liste;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    select DISTINCT groupe.grp_id, groupe.grp_nom, eta_id, eta_nom
    FROM groupe
    INNER JOIN etablisement USING(eta_id)
    INNER JOIN groupe_secteur USING (grp_id)
    INNER JOIN document.documents_secteur USING (sec_id)
    where documents_secteur.dos_id = prm_dos_id
    ORDER BY eta_nom, grp_nom
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

document\_documents\_liste — Retourne la liste des configurations de pages de documents.

## Synopsis

```
setof documents document_documents_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des configurations de pages de documents. Entrées :

- `prm_token` : Token d'authentification

Remarques : Nécessite les droits de configuration de l'interface.

## Source

```
DECLARE
    row document.documents;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row IN
        SELECT * FROM document.documents ORDER BY dos_titre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

document\_documents\_liste\_details

## Synopsis

```
setof                               document_documents_liste_details  
document_documents_liste_details(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE  
  row document.documents_liste_details;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT  
      dos_id,  
      dos_titre,  
      concatenate (DISTINCT secteur.sec_nom),  
      dty_nom  
    FROM document.documents  
    LEFT JOIN document.documents_secteur USING(dos_id)  
    LEFT JOIN meta.secteur USING(sec_id)  
    LEFT JOIN document.document_type USING(dty_id)  
    GROUP BY dos_id, dos_titre, dty_nom  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

document\_documents\_save — Modifie les informations de configuration d'une page de documents ou crée une nouvelle configuration.

## Synopsis

```
int4 document_documents_save(prm_token, prm_dos_id, prm_titre, prm_dty_id);
```

```
int4 prm_token;  
int4 prm_dos_id;  
varchar prm_titre;  
int4 prm_dty_id;
```

## Description

Modifie les informations de configuration d'une page de documents ou crée une nouvelle configuration. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dos\_id* : Identifiant de la configuration de page à modifier ou NULL pour créer une nouvelle configuration
- *prm\_titre* : Nouveau nom de page
- *prm\_dty\_id* : Identifiant du type de document permettant de filtrer les documents sur cette page

Remarque : Nécessite le droit de configuration de l'interface

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  IF prm_dos_id ISNULL THEN  
    INSERT INTO document.documents (dos_titre, dty_id) VALUES (prm_titre, prm_dty_id)  
    RETURNING dos_id INTO ret;  
    RETURN ret;  
  ELSE  
    UPDATE document.documents SET dos_titre = prm_titre, dty_id = prm_dty_id WHERE dos_id = prm_dos_id;  
    RETURN prm_dos_id;  
  END IF;  
END;
```

## Name

document\_documents\_secteur\_ajoute — Ajoute un secteur à la spécialisation de la page de documents par secteur.

## Synopsis

```
int4 document_documents_secteur_ajoute(prm_token, prm_dos_id, prm_sec_id);
```

```
int4 prm_token;
```

```
int4 prm_dos_id;
```

```
int4 prm_sec_id;
```

## Description

Ajoute un secteur à la spécialisation de la page de documents par secteur. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dos\_id* : Identifiant de la configuration de page de documents
- *prm\_sec\_id* : Identifiant du secteur

Remarques : Nécessite les droits de configuration de l'interface.

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  INSERT INTO document.documents_secteur (dos_id, sec_id) VALUES (prm_dos_id, prm_sec_id)
  RETURNING dss_id INTO ret;
  RETURN ret;
END;
```

## Name

`document_documents_secteur_liste_details_etab` — Retourne la liste des secteurs sur lesquels est spécialisée une page de documents.

## Synopsis

```
setof                               document_documents_secteur_liste_details
document_documents_secteur_liste_details_etab(prm_token,           prm_dos_id,
prm_eta_id);
```

```
int4 prm_token;
int4 prm_dos_id;
int4 prm_eta_id;
```

## Description

Retourne la liste des secteurs sur lesquels est spécialisée une page de documents. Entrées :

- `prm_token` : Token d'authentification
- `prm_dos_id` : Identifiant de la configuration de page de documents
- `prm_eta_id` : Identifiant de l'établissement sur lequel filtrer les secteurs

## Source

```
DECLARE
    row document.document_documents_secteur_liste_details;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT dss_id, sec_id, sec_nom, sec_icone
        FROM document.documents_secteur
        INNER JOIN meta.secteur USING(sec_id)
        LEFT JOIN etablisement_secteur USING(sec_id)
        WHERE dos_id = prm_dos_id AND (prm_eta_id ISNULL OR eta_id = prm_eta_id)
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```



## Name

document\_documents\_secteur\_supprime — Enlève un secteur à la spécialisation de la page de documents par secteur.

## Synopsis

```
void document_documents_secteur_supprime(prm_token, prm_dss_id);
```

```
int4 prm_token;
```

```
int4 prm_dss_id;
```

## Description

Enlève un secteur à la spécialisation de la page de documents par secteur. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dss\_id* : Identifiant de la spécialisation de la page de documents par un secteur

Remarques : Nécessite les droits de configuration de l'interface.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM document.documents_secteur WHERE dss_id = prm_dss_id;
END;
```

## Name

document\_documents\_secteurs\_set — Indique les secteurs sur lesquels est spécialisée une vue de documents.

## Synopsis

```
void document_documents_secteurs_set(prm_token, prm_dos_id, prm_sec_codes);
```

```
int4 prm_token;
```

```
int4 prm_dos_id;
```

```
varchar[] prm_sec_codes;
```

## Description

Indique les secteurs sur lesquels est spécialisée une vue de documents.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dos\_id* : Identifiant de la vue de documents
- *prm\_sec\_codes* : tableau de codes de secteurs

Remarques :

- Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  DELETE FROM document.documents_secteur WHERE dos_id = prm_dos_id;
  IF prm_sec_codes NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_sec_codes, 1) LOOP
      INSERT INTO document.documents_secteur (dos_id, sec_id) VALUES (prm_dos_id, (SELECT sec_id FR
    END LOOP;
  END IF;
END;
```

## Name

document\_documents\_supprime — Supprime une configuration de page de documents.

## Synopsis

```
void document_documents_supprime(prm_token, prm_dos_id);
```

```
int4 prm_token;  
int4 prm_dos_id;
```

## Description

Supprime une configuration de page de documents. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_dos\_id* : Identifiant de la configuration de page de documents

Remarque : Nécessite le droit de configuration de l'interface

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  DELETE FROM document.documents_secteur WHERE dos_id = prm_dos_id;  
  DELETE FROM document.documents WHERE dos_id = prm_dos_id;  
END;
```

## 5. events

### 5.1. Description

Données des événements rattachés aux usagers. Il est possible de définir des pages événements, spécialisables par secteur et par catégories d'événements. Il est possible de créer des événements et de les rattacher à des personnes (usager, personnel, contact, famille) et des secteurs. On peut lui spécifier un type d'événement (d'où découle une catégorie), et y affecter des ressources (voir schéma ressource). Il est possible de définir des types d'événements (classifiés en catégories) et de les affecter à des secteurs. Chacun de ces types d'événements pourra être utilisé ou non par chacun des établissements du réseau. Il est possible de définir des pages Agenda ressource, résumant l'utilisation des ressources affectées aux événements.

### 5.2. Tables

#### 5.2.1. events.agressources

Configuration des pages d'agenda de ressources

*agr\_id* (int4)

*agr\_code* (varchar)

*agr\_titre* (varchar)

#### 5.2.2. events.agressources\_secteur

Spécialisation des pages d'agenda par secteur

*ase\_id* (int4)

agr\_id (int4)

sec\_id (int4)

### 5.2.3. events.categorie\_events

Spécialisation des pages d'événements à certaines catégories d'événements

cev\_id (int4)

eca\_id (int4)

Clé étrangère vers events\_categorie.eca\_id

evs\_id (int4)

Clé étrangère vers events.evs\_id

### 5.2.4. events.event

Les événements. Un événement est caractérisé par un type d'événement (les types étant eux-mêmes classifiés en catégories) et une période de temps. La période est expérimentée soit par un intervalle entre 2 heures précises, soit par journées entières, soit par une heure précise pour un événement ponctuel. Certains champs sont spécifiques aux secteurs associés à l'événement.

eve\_id (int4)

eve\_intitule (varchar)

Intitulé de l'événement

eve\_debut (timestampz)

Date de début de l'événement. Si l'événement est de type journée, l'heure doit être 00:00

eve\_fin (timestampz)

Date de fin de l'événement. Si l'événement est de type journée, l'heure doit être 00:00 et l'événement termine à la fin de la journée indiquée. Si l'événement est de type ponctuel, cette valeur doit être égale à celle de eve\_debut.

eve\_\_depenses\_montant (numeric)

Si le secteur depenses est associé à l'événement, ce champs indique le montant de la dépense associée à l'événement

uti\_id\_creation (int4)

Clé étrangère vers utilisateur.uti\_id

eca\_id (int4)

Identifiant de la catégorie d'événement (découle directement du type donné par ety\_id)

ety\_id (int4)

Clé étrangère vers event\_type.ety\_id

Identifiant du type d'événement

eve\_date\_creation (timestampz)

eve\_journee (bool)

Indique s'il s'agit d'un événement sur une ou des journées entières.

eve\_ponctuel (bool)

Indique s'il s'agit d'un événement à une heure précise, sans durée particulière.

## Liens vers cette table

- event\_memo.eve\_id
- event\_personne.eve\_id
- event\_ressource.eve\_id
- secteur\_event.eve\_id

### 5.2.5. events.event\_memo

Textes accompagnant les événements (objet, compte-rendu)

eme\_id (int4)

eve\_id (int4)

Clé étrangère vers event.eve\_id

eme\_timestamp (timestampz)

eme\_type (varchar)

Type de mémo : bilan ou description

eme\_texte (text)

### 5.2.6. events.event\_personne

Rattachement de personnes (usager, personnel, contact, famille) aux événements

epe\_id (int4)

eve\_id (int4)

Clé étrangère vers event.eve\_id

per\_id (int4)

Clé étrangère vers personne.per\_id

### 5.2.7. events.event\_ressource

Affectation de ressources aux événements

ere\_id (int4)

eve\_id (int4)

Clé étrangère vers event.eve\_id

res\_id (int4)

Clé étrangère vers ressource.res\_id

### 5.2.8. events.event\_type

Types d'événements (sous-niveau de catégories d'événements)

ety\_id (int4)

eca\_id (int4)

Clé étrangère vers events\_categorie.eca\_id

ety\_intitule (varchar)

ety\_intitule\_individuel (bool)

### **Liens vers cette table**

- event.ety\_id
- event\_type\_etablissement.ety\_id
- event\_type\_secteur.ety\_id
- events.ety\_id

### **5.2.9. events.event\_type\_etablissement**

Utilisation d'un type d'événement par un établissement

ete\_id (int4)

ety\_id (int4)

Clé étrangère vers event\_type.ety\_id

eta\_id (int4)

Clé étrangère vers etablissement.eta\_id

### **5.2.10. events.event\_type\_secteur**

Affectation d'un type d'événement à des secteurs

ets\_id (int4)

ety\_id (int4)

Clé étrangère vers event\_type.ety\_id

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

### **5.2.11. events.events**

Configuration des pages d'événements disponibles pour placer dans le menu principal ou usager

evs\_id (int4)

evs\_titre (varchar)

evs\_code (varchar)

ety\_id (int4)

Clé étrangère vers event\_type.ety\_id

### **Liens vers cette table**

- categorie\_events.evs\_id
- secteur\_events.evs\_id

### **5.2.12. events.events\_categorie**

Catégories d'événements

eca\_id (int4)

eca\_nom (varchar)

eca\_code (varchar)

eca\_icone (varchar)

#### **Liens vers cette table**

- categorie\_events.eca\_id
- event\_type.eca\_id

### **5.2.13. events.secteur\_event**

Affectation des événements à des secteurs

set\_id (int4)

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

eve\_id (int4)

Clé étrangère vers event.eve\_id

### **5.2.14. events.secteur\_events**

Spécialisation des pages d'événements à certains secteurs

sev\_id (int4)

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

evs\_id (int4)

Clé étrangère vers events.evs\_id

## **5.3. Types**

### **5.3.1. events.events\_document\_liste**

### **5.3.2. events.events\_echeance\_liste**

### **5.3.3. events.events\_event\_liste**

### **5.3.4. events.events\_event\_type\_list\_all**

### **5.3.5. events.events\_event\_type\_list\_par\_evs**

### **5.3.6. events.events\_groupe\_liste**

### **5.3.7. events.events\_groupe\_liste2**

### **5.3.8. events.events\_agressources\_list\_details**

### **5.3.9. events.events\_event\_avec\_ressource\_liste**

### **5.3.10. events.events\_event\_bilan**

- nombre : nombre d'événements du type spécifié - duree\_heures : nombre d'heures couvertes par des événements (uniquement les événements normaux : ni toute la journée, ni ponctuels) - duree\_jours : nombre de jours couverts par les événements (uniquement pour les événements toute la journée) - eca\_nom : nom de la catégorie du type - ety\_id : identifiant du type - ety\_intitule : intitulé du type

### **5.3.11. events.events\_events\_liste\_details**

## **5.4. Fonctions**



## Name

events\_agressources\_get — Retourne les informations de configuration d'une page agenda de ressources.

## Synopsis

```
agressources events_agressources_get(prm_token, prm_agr_id);
```

```
int4 prm_token;  
int4 prm_agr_id;
```

## Description

Retourne les informations de configuration d'une page agenda de ressources. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_agr\_id* : Identifiant de la configuration de page agenda de ressources

## Source

```
DECLARE  
    ret events.agressources;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM events.agressources WHERE agr_id = prm_agr_id;  
    RETURN ret;  
END;
```

## Name

events\_agressources\_get\_par\_code — Retourne les informations de configuration d'une page agenda de ressources.

## Synopsis

```
agressources events_agressources_get_par_code(prm_token, prm_agr_code);
```

```
int4 prm_token;
```

```
varchar prm_agr_code;
```

## Description

Retourne les informations de configuration d'une page agenda de ressources. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_agr\_code* : Code de la configuration de page agenda de ressources

## Source

```
DECLARE
    ret events.agressources;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM events.agressources WHERE agr_code = prm_agr_code;
    RETURN ret;
END;
```

## Name

events\_agressources\_list — Retourne la liste des informations de configuration de pages d'agenda de ressources, à placer dans le menu principal ou usager.

## Synopsis

```
setof agressources events_agressources_list(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des informations de configuration de pages d'agenda de ressources, à placer dans le menu principal ou usager. Remarque : Nécessite le droit de configuration de l'interface

## Source

```
DECLARE
    row events.agressources;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row IN
        SELECT * FROM events.agressources ORDER BY agr_titre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_agressources\_list\_details — Retourne la liste des informations de configuration de pages d'agenda de ressources, à placer dans le menu principal ou usager.

## Synopsis

```
setof          events_agressources_list_details
events_agressources_list_details(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des informations de configuration de pages d'agenda de ressources, à placer dans le menu principal ou usager.

## Source

```
DECLARE
  row events.events_agressources_list_details;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT
      agr_id,
      agr_code,
      agr_titre,
      concatenate (DISTINCT secteur.sec_nom)
    FROM events.agressources
    LEFT JOIN events.agressources_secteur USING(agr_id)
    LEFT JOIN meta.secteur USING(sec_id)
    GROUP BY agr_id, agr_code, agr_titre
    ORDER BY agr_titre
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

events\_agressources\_save — Modifie les informations d'une vue de ressources ou crée une nouvelle vue.

## Synopsis

```
int4 events_agressources_save(prm_token, prm_agr_id, prm_code, prm_titre);
```

```
int4 prm_token;
```

```
int4 prm_agr_id;
```

```
varchar prm_code;
```

```
varchar prm_titre;
```

## Description

Modifie les informations d'une vue de ressources ou crée une nouvelle vue. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_agr\_id* : Identifiant de la vue à modifier ou NULL pour créer une nouvelle vue
- *prm\_titre* : Nouveau nom de vue
- *prm\_code* : Nouveau code de vue

## Source

```
DECLARE
    ret integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    IF prm_agr_id ISNULL THEN
        INSERT INTO events.agressources (agr_titre, agr_code) VALUES (prm_titre, prm_code)
            RETURNING agr_id INTO ret;
        RETURN ret;
    ELSE
        UPDATE events.agressources SET agr_titre = prm_titre, agr_code = prm_code WHERE agr_id = prm_agr_id;
        RETURN prm_agr_id;
    END IF;
END;
```

## Name

events\_agressources\_secteur\_liste — Retourne la liste des secteurs sur lesquels une vue de ressources est spécialisée.

## Synopsis

```
setof secteur events_agressources_secteur_liste(prm_token, prm_agr_id);
```

```
int4 prm_token;  
int4 prm_agr_id;
```

## Description

Retourne la liste des secteurs sur lesquels une vue de ressources est spécialisée.

## Source

```
DECLARE  
    row meta.secteur;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN  
        SELECT secteur.* FROM meta.secteur  
            INNER JOIN events.agressources_secteur USING (sec_id)  
            WHERE agr_id = prm_agr_id  
            ORDER BY sec_nom  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

events\_agressources\_secteurs\_set — Indique les secteurs sur lesquels est spécialisée une vue de ressources.

## Synopsis

```
void events_agressources_secteurs_set(prm_token, prm_agr_id, prm_secteurs);
```

```
int4 prm_token;  
int4 prm_agr_id;  
varchar[] prm_secteurs;
```

## Description

Indique les secteurs sur lesquels est spécialisée une vue de ressources.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_agr\_id* : Identifiant de la vue de ressources
- *prm\_secteurs* : Tableau de codes de secteurs

Remarques :

- Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  DELETE FROM events.agressources_secteur WHERE agr_id = prm_agr_id;  
  IF prm_secteurs NOTNULL THEN  
    FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP  
      INSERT INTO events.agressources_secteur (agr_id, sec_id) VALUES (prm_agr_id, (SELECT sec_id FROM me  
    END LOOP;  
  END IF;  
END;
```

## Name

events\_agressources\_supprime — Supprime une vue de ressources

## Synopsis

```
void events_agressources_supprime(prm_token, prm_agr_id);
```

```
int4 prm_token;
```

```
int4 prm_agr_id;
```

## Description

Supprime une vue de ressources

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  DELETE FROM events.agressources_secteur WHERE agr_id = prm_agr_id;
  DELETE FROM events.agressources WHERE agr_id = prm_agr_id;
END;
```



## Name

events\_categorie\_events\_liste — Retourne la liste des catégories d'événements sur lesquelles une page d'événements est spécialisée.

## Synopsis

```
setof events_categorie events_categorie_events_liste(prm_token, prm_evs_id);
```

```
int4 prm_token;  
int4 prm_evs_id;
```

## Description

Retourne la liste des catégories d'événements sur lesquelles une page d'événements est spécialisée. Entrées :

- prm\_token : Token d'authentification
- prm\_evs\_id : Identifiant de la configuration de page d'événement

## Source

```
DECLARE  
    row events.events_categorie;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN  
        SELECT DISTINCT events_categorie.* FROM events.events_categorie  
        INNER JOIN events.categorie_events USING (eca_id)  
        WHERE evs_id = prm_evs_id  
        ORDER BY eca_nom  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

events\_categorie\_events\_set — Indique les catégories d'événements sur lesquels est spécialisée une vue d'événements.

## Synopsis

```
void events_categorie_events_set(prm_token, prm_evs_id, prm_ecas);
```

```
int4 prm_token;
```

```
int4 prm_evs_id;
```

```
varchar[] prm_ecas;
```

## Description

Indique les catégories d'événements sur lesquels est spécialisée une vue d'événements.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identifiant de la vue d'événements
- *prm\_ecas* : Tableau de codes de catégories d'événements

Remarques :

- Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  DELETE FROM events.categorie_events WHERE evs_id = prm_evs_id;
  IF prm_ecas NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_ecas, 1) LOOP
      INSERT INTO events.categorie_events (evs_id, eca_id) VALUES (prm_evs_id, (SELECT eca_id FROM events
      END LOOP;
  END IF;
END;
```

## Name

events\_document\_liste\_obtention — Retourne sous forme d'événement la liste des documents dont la date d'obtention est comprise dans une période donnée.

## Synopsis

```
setof events_document_liste events_document_liste_obtention(prm_token,
prm_evs_id, prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne sous forme d'événement la liste des documents dont la date d'obtention est comprise dans une période donnée. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identification de la configuration de page d'événement (pour trier sur les documents du même secteur que la page)
- *prm\_per\_id* : Identifiant d'un usager (pour spécialiser la recherche aux documents rattachés à cet usager) ou NULL
- *prm\_start* : Date de début de période de recherche
- *prm\_end* : Date de fin de période de recherche
- *prm\_grp\_id* : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux documents rattachés aux utilisateurs de ce groupe ou NULL
- *prm\_per\_ids* : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche aux documents rattachés à un de ces utilisateurs au moins

## Source

```
DECLARE
  row events.events_document_liste;
  p_start timestamp;
  p_end timestamp;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  if prm_start NOTNULL THEN
    p_start = prm_start;
  ELSE
    p_start = timestamp '-INFINITY';
  END IF;
  if prm_end NOTNULL THEN
    p_end = prm_end;
  ELSE
    p_end = timestamp 'INFINITY';
  END IF;
  FOR row IN
    SELECT DISTINCT doc_id, doc_titre, doc_description, EXTRACT (EPOCH FROM doc_date_obtention),
      (SELECT sec_icone FROM meta.secteur INNER JOIN document.document_secteur USING(sec_id) WHERE document.
      (SELECT concatenate(personne_get_libelle_initiale(prm_token, per_id_usager)) FROM document.document_u
```

```
FROM document.document
INNER JOIN document.document_usager USING(doc_id)
INNER JOIN document.document_secteur USING(doc_id)
INNER JOIN events.secteur_events USING(sec_id)
INNER JOIN personne_groupe on personne_groupe.per_id = document_usager.per_id_usager AND
    doc_date_obtention BETWEEN COALESCE(personne_groupe.peg_debut, '-Infinity'::timestamp) AND COALESCE
WHERE
    secteur_events.evs_id = prm_evs_id
    AND doc_date_obtention between p_start AND p_end
    AND (prm_per_id ISNULL OR prm_per_id = document_usager.per_id_usager)
    AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)
    AND (prm_per_ids ISNULL OR document_usager.per_id_usager = ANY (prm_per_ids))
LOOP
    RETURN NEXT row;
END LOOP;
END;
```

## Name

events\_document\_liste\_realisation — Retourne sous forme d'événement la liste des documents dont la date de réalisation est comprise dans une période donnée.

## Synopsis

```
setof events_document_liste events_document_liste_realisation(prm_token,
prm_evs_id, prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne sous forme d'événement la liste des documents dont la date de réalisation est comprise dans une période donnée. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identification de la configuration de page d'événement (pour trier sur les documents du même secteur que la page)
- *prm\_per\_id* : Identifiant d'un usager (pour spécialiser la recherche aux documents rattachés à cet usager) ou NULL
- *prm\_start* : Date de début de période de recherche
- *prm\_end* : Date de fin de période de recherche
- *prm\_grp\_id* : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux documents rattachés aux utilisateurs de ce groupe ou NULL
- *prm\_per\_ids* : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche aux documents rattachés à un de ces utilisateurs au moins

## Source

```
DECLARE
  row events.events_document_liste;
  p_start timestamp;
  p_end timestamp;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  if prm_start NOTNULL THEN
    p_start = prm_start;
  ELSE
    p_start = timestamp '-INFINITY';
  END IF;
  if prm_end NOTNULL THEN
    p_end = prm_end;
  ELSE
    p_end = timestamp 'INFINITY';
  END IF;

  FOR row IN
    SELECT DISTINCT doc_id, doc_titre, doc_description, EXTRACT (EPOCH FROM doc_date_realisation),
      (SELECT sec_icone FROM meta.secteur INNER JOIN document.document_secteur USING(sec_id) WHERE document
```

```
(SELECT concatenate(personne_get_libelle_initiale(prm_token, per_id_usager)) FROM document.document_u
FROM document.document
INNER JOIN document.document_usager USING(doc_id)
INNER JOIN document.document_secteur USING(doc_id)
INNER JOIN events.secteur_events USING(sec_id)
INNER JOIN personne_groupe on personne_groupe.per_id = document_usager.per_id_usager AND
    doc_date_realisation BETWEEN COALESCE(personne_groupe.peg_debut, '-Infinity'::timestamp) AND COALES
WHERE
    secteur_events.evs_id = prm_evs_id
    AND doc_date_realisation between p_start AND p_end
    AND (prm_per_id ISNULL OR prm_per_id = document_usager.per_id_usager)
    AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)
    AND (prm_per_ids ISNULL OR document_usager.per_id_usager = ANY(prm_per_ids))
LOOP
    RETURN NEXT row;
END LOOP;
END;
```

## Name

events\_document\_liste\_validite — Retourne sous forme d'événement la liste des documents dont la date de validité est comprise dans une période donnée.

## Synopsis

```
setof events_document_liste events_document_liste_validite(prm_token,
prm_evs_id, prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne sous forme d'événement la liste des documents dont la date de validité est comprise dans une période donnée. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identification de la configuration de page d'événement (pour trier sur les documents du même secteur que la page)
- *prm\_per\_id* : Identifiant d'un usager (pour spécialiser la recherche aux documents rattachés à cet usager) ou NULL
- *prm\_start* : Date de début de période de recherche
- *prm\_end* : Date de fin de période de recherche
- *prm\_grp\_id* : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux documents rattachés aux utilisateurs de ce groupe ou NULL
- *prm\_per\_ids* : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche aux documents rattachés à un de ces utilisateurs au moins

## Source

```
DECLARE
  row events.events_document_liste;
  p_start timestamp;
  p_end timestamp;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  if prm_start NOTNULL THEN
    p_start = prm_start;
  ELSE
    p_start = timestamp '-INFINITY';
  END IF;
  if prm_end NOTNULL THEN
    p_end = prm_end;
  ELSE
    p_end = timestamp 'INFINITY';
  END IF;
  FOR row IN
    SELECT DISTINCT doc_id, doc_titre, doc_description, EXTRACT (EPOCH FROM doc_date_validite),
      (SELECT sec_icone FROM meta.secteur INNER JOIN document.document_secteur USING(sec_id) WHERE document.
      (SELECT concatenate(personne_get_libelle_initiale(prm_token, per_id_usager)) FROM document.document_u
```

```
FROM document.document
INNER JOIN document.document_usager USING(doc_id)
INNER JOIN document.document_secteur USING(doc_id)
INNER JOIN events.secteur_events USING(sec_id)
INNER JOIN personne_groupe on personne_groupe.per_id = document_usager.per_id_usager AND
    doc_date_validite BETWEEN COALESCE(personne_groupe.peg_debut, '-Infinity'::timestamp) AND COALESCE
WHERE
    secteur_events.evs_id = prm_evs_id
    AND doc_date_validite between p_start AND p_end
    AND (prm_per_id ISNULL OR prm_per_id = document_usager.per_id_usager)
    AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)
    AND (prm_per_ids ISNULL OR document_usager.per_id_usager = ANY(prm_per_ids))
LOOP
    RETURN NEXT row;
END LOOP;
END;
```



## Name

events\_echeance\_liste — Retourne sous forme d'événement la liste des échéances arrivées à terme lors d'une période donnée.

## Synopsis

```
setof events_echeance_liste events_echeance_liste(prm_token, prm_evs_id,
prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne sous forme d'événement la liste des échéances arrivées à terme lors d'une période donnée. Une échéance est définie par un champ usager de format date marquée comme échéance. Entrées :

- prm\_token : Token d'authentification
- prm\_evs\_id : Identification de la configuration de page d'événement (pour trier sur les champs échéance du même secteur que la page)
- prm\_per\_id : Identifiant d'un usager (pour spécialiser la recherche aux échéances de cet usager) ou NULL
- prm\_start : Date de début de période de recherche
- prm\_end : Date de fin de période de recherche
- prm\_grp\_id : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux échéances des utilisateurs de ce groupe ou NULL
- prm\_per\_ids : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche aux échéances d'un de ces utilisateurs au moins

## Source

```
DECLARE
row events_echeance_liste;
p_start timestamp;
p_end timestamp;
BEGIN
PERFORM login._token_assert (prm_token, FALSE, FALSE);
if prm_start NOTNULL THEN
p_start = prm_start;
ELSE
p_start = timestamp '-INFINITY';
END IF;
if prm_end NOTNULL THEN
p_end = prm_end;
ELSE
p_end = timestamp 'INFINITY';
END IF;
FOR row IN
SELECT DISTINCT
personne_info.per_id,
info.inf_id,
inf_libelle,
inf_libelle_complet,
```

```
personne_info_varchar_get (prm_token, personne_info.per_id, 'nom'),
personne_info_varchar_get (prm_token, personne_info.per_id, 'prenom'),
EXTRACT (EPOCH FROM personne_info_date_get (prm_token, personne_info.per_id, inf_code) ),
inf_date_echeance_icone
FROM personne_info_date
INNER JOIN personne_info USING (pin_id)
INNER JOIN meta.info USING(inf_code)
INNER JOIN meta.secteur ON info.inf__date_echeance_secteur = secteur.sec_code
INNER JOIN events.secteur_events USING(sec_id)
INNER JOIN personne_groupe ON personne_groupe.per_id = personne_info.per_id
    AND personne_info_date_get (prm_token, personne_info.per_id, inf_code) BETWEEN COALESCE (personne
WHERE inf__date_echeance
AND pid_valeur BETWEEN p_start AND p_end
AND (prm_per_id ISNULL OR personne_info.per_id = prm_per_id)
AND evs_id = prm_evs_id
AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)
AND (prm_per_ids ISNULL OR personne_info.per_id = ANY(prm_per_ids))
LOOP
RETURN NEXT row;
END LOOP;
END;
```

## Name

events\_event\_avec\_ressource\_liste — Retourne sous forme d'événement la liste des ressources utilisées lors d'événements

## Synopsis

```
setof          events_event_avec_ressource_liste
events_event_avec_ressource_liste(prm_token,      prm_start,      prm_end,
prm_agr_id);
```

```
int4 prm_token;
date prm_start;
date prm_end;
int4 prm_agr_id;
```

## Description

Retourne sous forme d'événement la liste des ressources utilisées lors d'événements Entrées :

- *prm\_token* : Token d'authentification
- *prm\_start* : Date de début de période de recherche
- *prm\_end* : Date de fin de période de recherche
- *prm\_agr\_id* : Identification de la configuration de page d'agenda de ressources (pour trier sur les ressources du même secteur que la page)

## Source

```
DECLARE
    row events.event_avec_ressource_liste;
    p_start timestamp;
    p_end timestamp;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    if prm_start NOTNULL THEN
        p_start = prm_start;
    ELSE
        p_start = timestamp '-INFINITY';
    END IF;
    if prm_end NOTNULL THEN
        p_end = prm_end;
    ELSE
        p_end = timestamp 'INFINITY';
    END IF;
    FOR row IN
        SELECT DISTINCT eve_id, res_id, res_nom, eve_intitule, EXTRACT(EPOCH FROM eve_debut), EXTRACT(EPOCH
        FROM events.event_avec_ressource_liste)
        FROM events.event
        inner join events.event_ressource using(eve_id)
        inner join ressource.ressource_secteur using(res_id)
        inner join ressource.ressource USING(res_id)
        inner join events.agressources_secteur using(sec_id)
        WHERE agr_id = prm_agr_id AND
        (eve_debut BETWEEN p_start AND p_end OR eve_fin BETWEEN p_start AND p_end)
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_event\_bilan

## Synopsis

```
setof events_event_bilan events_event_bilan(prm_token, prm_evs_id,
prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne un bilan des événements sur une période donnée, regroupés par type d'événement.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identification de la configuration de page d'événement (pour trier sur les événements du même secteur que la page)
- *prm\_per\_id* : Identifiant d'un usager (pour spécialiser la recherche aux événements liés à cet usager) ou NULL
- *prm\_start* : Date de début de période de recherche
- *prm\_end* : Date de fin de période de recherche
- *prm\_grp\_id* : Identifiant d'un groupe d'usagers, pour spécialiser la recherche aux événements liés aux usagers de ce groupe ou NULL
- *prm\_per\_ids* : Tableau d'identifiants d'usagers, pour spécialiser la recherche aux événements liés à un de ces usagers au moins

## Source

```
DECLARE
  row events_event_bilan;
  p_start timestamp;
  p_end timestamp;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  if prm_start NOTNULL THEN
    p_start = prm_start;
  ELSE
    p_start = timestamp '-INFINITY';
  END IF;
  if prm_end NOTNULL THEN
    p_end = prm_end;
  ELSE
    p_end = timestamp 'INFINITY';
  END IF;
  FOR row IN
    select count(*),
           SUM(heures),
           SUM(journees),
           sub.eca_nom,
           sub.ety_id,
```

```

sub.ety_intitule
  from (
    SELECT DISTINCT
      eve_id,
      (CASE WHEN eve_journee THEN 0 ELSE 1 END)*EXTRACT(epoch FROM age(eve_fin, eve_debut)/3600) AS he
      (CASE WHEN eve_journee THEN 1 ELSE 0 END)*(1+EXTRACT(epoch FROM age(eve_fin, eve_debut)/3600/24)
        event.eca_id,
        eca_nom,
        event.ety_id,
        ety_intitule
    FROM events.event
    INNER JOIN events.event_type_secteur USING (ety_id)
    INNER JOIN events.secteur_events ON event_type_secteur.sec_id = secteur_events.sec_id
    INNER JOIN events.events USING(evs_id)
    INNER JOIN events.categorie_events USING (evs_id)
    LEFT JOIN events.events_categorie ON event.eca_id = events_categorie.eca_id
    INNER JOIN events.event_personne USING(eve_id)
    LEFT JOIN personne_groupe ON personne_groupe.per_id = event_personne.per_id AND (COALESC
    LEFT JOIN events.event_type ON event.ety_id=event_type.ety_id
    WHERE
      secteur_events.evs_id = prm_evs_id AND categorie_events.eca_id = event.eca_id
      AND (prm_per_id ISNULL OR event_personne.per_id = prm_per_id)
    AND (eve_debut BETWEEN p_start AND p_end OR eve_fin BETWEEN p_start AND p_end)
      AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)
      AND (prm_per_ids ISNULL OR event_personne.per_id = ANY(prm_per_ids))
      AND (events.ety_id ISNULL OR events.ety_id = event.ety_id)) sub
    GROUP BY
      eca_nom,
      ety_id,
      ety_intitule
  LOOP
    RETURN NEXT row;
  END LOOP;
END;

```

## Name

events\_event\_get — Retourne les informations sur un événement.

## Synopsis

```
event events_event_get(prm_token, prm_eve_id);
```

```
int4 prm_token;  
int4 prm_eve_id;
```

## Description

Retourne les informations sur un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement

## Source

```
DECLARE  
    ret events.event;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM events.event WHERE eve_id = prm_eve_id;  
    RETURN ret;  
END;
```

## Name

events\_event\_liste — Retourne la liste des événements d'une période donnée.

## Synopsis

```
setof events_event_liste events_event_liste(prm_token, prm_evs_id,
prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne la liste des événements d'une période donnée. Entrées :

- prm\_token : Token d'authentification
- prm\_evs\_id : Identification de la configuration de page d'événement (pour trier sur les événements du même secteur que la page)
- prm\_per\_id : Identifiant d'un usager (pour spécialiser la recherche aux événements liés à cet usager) ou NULL
- prm\_start : Date de début de période de recherche
- prm\_end : Date de fin de période de recherche
- prm\_grp\_id : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux événements liés aux utilisateurs de ce groupe ou NULL
- prm\_per\_ids : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche aux événements liés à un de ces utilisateurs au moins

## Source

```
DECLARE
row events_event_liste;
p_start timestamp;
p_end timestamp;
BEGIN
PERFORM login._token_assert (prm_token, FALSE, FALSE);
set enable_seqscan to false;
if prm_start NOTNULL THEN
p_start = prm_start;
ELSE
p_start = timestamp '-INFINITY';
END IF;
if prm_end NOTNULL THEN
p_end = prm_end;
ELSE
p_end = timestamp 'INFINITY';
END IF;
-- RAISE NOTICE '% %', p_start, p_end;
FOR row IN
SELECT DISTINCT
event.eve_id,
event.eca_id,
eve_intitule,
eve_journee,
```

```
eve_ponctuel,  
EXTRACT(EPOCH FROM eve_debut),  
EXTRACT(EPOCH FROM eve_fin),  
CASE WHEN eca_icone NOTNULL THEN eca_icone ELSE (SELECT sec_icone FROM meta.secteur INNER JOIN even  
eve_depenses_montant  
FROM events.event  
INNER JOIN events.event_type_secteur USING (ety_id)  
INNER JOIN events.secteur_events ON event_type_secteur.sec_id = secteur_events.sec_id  
INNER JOIN events.events USING(evs_id)  
INNER JOIN events.categorie_events USING (evs_id)  
LEFT JOIN events.events_categorie ON event.eca_id = events_categorie.eca_id  
INNER JOIN events.event_personne USING(eve_id)  
LEFT JOIN personne_groupe ON personne_groupe.per_id = event_personne.per_id AND (COALESCE (personne_g  
WHERE secteur_events.evs_id = prm_evs_id AND categorie_events.eca_id = event.eca_id  
AND (prm_per_id ISNULL OR event_personne.per_id = prm_per_id) AND  
(eve_debut BETWEEN p_start AND p_end OR eve_fin BETWEEN p_start AND p_end)  
AND (prm_grp_id ISNULL OR prm_grp_id = personne_groupe.grp_id)  
AND (prm_per_ids ISNULL OR event_personne.per_id = ANY(prm_per_ids))  
AND (events.ety_id ISNULL OR events.ety_id = event.ety_id)  
LOOP  
RETURN NEXT row;  
END LOOP;  
set enable_seqscan to true;  
END;
```



## Name

events\_event\_memo\_get — Retourne un texte (objet, compte-rendu) d'un événement.

## Synopsis

```
text events_event_memo_get(prm_token, prm_eve_id, prm_type);
```

```
int4 prm_token;  
int4 prm_eve_id;  
varchar prm_type;
```

## Description

Retourne un texte (objet, compte-rendu) d'un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement
- *prm\_type* : Type du mémo : description (Objet) ou bilan (Compte-rendu)

## Source

```
DECLARE  
  ret text;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  SELECT eme_texte INTO ret FROM events.event_memo WHERE eve_id = prm_eve_id AND eme_type = prm_type ORDER BY  
  RETURN ret;  
END;
```

## Name

events\_event\_personne\_list — Retourne la liste des personnes rattachées à un événement.

## Synopsis

```
setof event_personne events_event_personne_list(prm_token, prm_eve_id);
```

```
int4 prm_token;
```

```
int4 prm_eve_id;
```

## Description

Retourne la liste des personnes rattachées à un événement.

## Source

```
DECLARE
    row events.event_personne;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM events.event_personne WHERE eve_id = prm_eve_id
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_event\_personnes\_save — Modifie les personnes rattachées à un événement.

## Synopsis

```
void events_event_personnes_save(prm_token, prm_eve_id, prm_per_ids);
```

```
int4 prm_token;  
int4 prm_eve_id;  
int4[] prm_per_ids;
```

## Description

Modifie les personnes rattachées à un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement
- *prm\_per\_ids* : Tableau d'identifiants de personnes à rattacher

## Source

```
DECLARE  
    row events.event_personne;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN  
        SELECT * FROM events.event_personne WHERE eve_id = prm_eve_id  
    LOOP  
        IF prm_per_ids ISNULL OR NOT row.per_id = ANY (prm_per_ids) THEN  
            DELETE FROM events.event_personne WHERE epe_id = row.epe_id;  
        END IF;  
    END LOOP;  
    IF prm_per_ids NOTNULL THEN  
        FOR i IN 1 .. array_upper (prm_per_ids, 1) LOOP  
            IF NOT EXISTS (SELECT 1 FROM events.event_personne WHERE eve_id = prm_eve_id AND per_id = prm_per_ids[i])  
                INSERT INTO events.event_personne (eve_id, per_id) VALUES (prm_eve_id, prm_per_ids[i]);  
            END IF;  
        END LOOP;  
    END IF;  
END;
```

## Name

events\_event\_ressource\_list — Retourne la liste des ressources affectées à un événement.

## Synopsis

```
setof ressource events_event_ressource_list(prm_token, prm_eve_id);
```

```
int4 prm_token;
```

```
int4 prm_eve_id;
```

## Description

Retourne la liste des ressources affectées à un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement

## Source

```
DECLARE
    row ressource.ressource;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT ressource.* FROM ressource.ressource INNER JOIN events.event_ressource USING(res_id) WHERE eve
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_event\_ressources\_save — Modifie les ressources rattachées à un événement.

## Synopsis

```
void events_event_ressources_save(prm_token, prm_eve_id, prm_res_ids);
```

```
int4 prm_token;
```

```
int4 prm_eve_id;
```

```
int4[] prm_res_ids;
```

## Description

Modifie les ressources rattachées à un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement
- *prm\_res\_ids* : Tableau d'identifiants de ressources à rattacher

## Source

```
DECLARE
    row events.event_ressource;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM events.event_ressource WHERE eve_id = prm_eve_id
    LOOP
        IF prm_res_ids ISNULL OR NOT row.res_id = ANY (prm_res_ids) THEN
            DELETE FROM events.event_ressource WHERE ere_id = row.ere_id;
        END IF;
    END LOOP;
    IF prm_res_ids NOTNULL THEN
        FOR i IN 1 .. array_upper (prm_res_ids, 1) LOOP
            IF NOT EXISTS (SELECT 1 FROM events.event_ressource WHERE eve_id = prm_eve_id AND res_id = prm_res_
                INSERT INTO events.event_ressource (eve_id, res_id) VALUES (prm_eve_id, prm_res_ids[i]);
            END IF;
        END LOOP;
    END IF;
END;
```

## Name

events\_event\_save\_all — Modifie un crée un événement.

## Synopsis

```
int4 events_event_save_all(prm_token, prm_eve_id, prm_intitule, prm_ety_id,  
prm_journee, prm_ponctuel, prm_debut, prm_fin, prm_depenses_montant,  
prm_description, prm_bilan, prm_per_ids, prm_res_ids, prm_sec_ids);
```

```
int4 prm_token;  
int4 prm_eve_id;  
varchar prm_intitule;  
int4 prm_ety_id;  
bool prm_journee;  
bool prm_ponctuel;  
timestampz prm_debut;  
timestampz prm_fin;  
numeric prm_depenses_montant;  
text prm_description;  
text prm_bilan;  
int4[] prm_per_ids;  
_int4 prm_res_ids;  
_int4 prm_sec_ids;
```

## Description

Modifie un crée un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement à modifier, ou NULL pour créer un nouvel événement
- *prm\_intitule* : Titre de l'événement
- *prm\_ety\_id* : Identifiant du type d'événement
- *prm\_journee* : Indique si l'événement couvre une ou plusieurs journées complètes
- *prm\_ponctuel* : Indique si l'événement est ponctuel (sans durée particulière)
- *prm\_debut* : Date de début de l'événement
- *prm\_fin* : Date de fin de l'événement
- *prm\_depenses\_montant* : Pour un événement de catégorie Dépenses, montant de la dépense
- *prm\_description* : Objet de l'événement
- *prm\_bilan* : Compte-rendu de l'événement
- *prm\_per\_ids* : Tableau d'identifiants de personnes liées à l'événement
- *prm\_res\_ids* : Tableau d'identifiants de ressources associées à l'événement
- *prm\_sec\_ids* : Tableau d'identifiants de secteurs auquel est affecté l'événement

## Source

```
DECLARE  
  new_eve_id integer;  
  uti integer;
```

```

BEGIN
PERFORM login._token_assert (prm_token, FALSE, FALSE);
SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
IF prm_eve_id NOTNULL THEN
  UPDATE events.event SET
    eve_intitule = prm_intitule,
    eca_id = (SELECT eca_id FROM events.event_type WHERE ety_id = prm_ety_id),
    ety_id = CASE WHEN prm_ety_id <> 0 THEN prm_ety_id ELSE NULL END,
    eve_journee = prm_journee,
    eve_ponctuel = prm_ponctuel,
    eve_debut = prm_debut,
    eve_fin = prm_fin,
    eve_depenses_montant = prm_depenses_montant
  WHERE eve_id = prm_eve_id;
INSERT INTO events.event_memo (eve_id, eme_timestamp, eme_type, eme_texte) VALUES (prm_eve_id, CURREN
INSERT INTO events.event_memo (eve_id, eme_timestamp, eme_type, eme_texte) VALUES (prm_eve_id, CURREN
new_eve_id = prm_eve_id;
ELSE
  INSERT INTO events.event (eve_intitule, eca_id, ety_id, eve_journee, eve_ponctuel, eve_debut, eve_fin
VALUES (prm_intitule, (SELECT eca_id FROM events.event_type WHERE ety_id = prm_ety_id), prm_ety_id, p
INSERT INTO events.event_memo (eve_id, eme_timestamp, eme_type, eme_texte) VALUES (new_eve_id, CURREN
INSERT INTO events.event_memo (eve_id, eme_timestamp, eme_type, eme_texte) VALUES (new_eve_id, CURREN
END IF;
IF prm_ponctuel AND prm_debut IS DISTINCT FROM prm_fin THEN
  UPDATE events.event SET eve_fin = eve_debut WHERE eve_id = new_eve_id;
END IF;
IF prm_per_ids <> '{0}' THEN
  PERFORM events.events_event_personnes_save (prm_token, new_eve_id, prm_per_ids);
ELSE
  PERFORM events.events_event_personnes_save (prm_token, new_eve_id, NULL);
END IF;
IF prm_res_ids <> '{0}' THEN
  PERFORM events.events_event_ressources_save (prm_token, new_eve_id, prm_res_ids);
ELSE
  PERFORM events.events_event_ressources_save (prm_token, new_eve_id, NULL);
END IF;
IF prm_sec_ids <> '{0}' THEN
  PERFORM events.events_event_secteurs_save (prm_token, new_eve_id, prm_sec_ids);
ELSE
  PERFORM events.events_event_secteurs_save (prm_token, new_eve_id, NULL);
END IF;
RETURN new_eve_id;
END;

```

## Name

events\_event\_secteurs\_save — Modifie les secteurs auxquels est affecté un événement.

## Synopsis

```
void events_event_secteurs_save(prm_token, prm_eve_id, prm_sec_ids);
```

```
int4 prm_token;
```

```
int4 prm_eve_id;
```

```
int4[] prm_sec_ids;
```

## Description

Modifie les secteurs auxquels est affecté un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement
- *prm\_sec\_ids* : Tableau d'identifiants de secteurs

## Source

```
DECLARE
    row events.secteur_event;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM events.secteur_event WHERE eve_id = prm_eve_id
    LOOP
        IF prm_sec_ids ISNULL OR NOT row.sec_id = ANY (prm_sec_ids) THEN
            DELETE FROM events.secteur_event WHERE set_id = row.set_id;
        END IF;
    END LOOP;
    IF prm_sec_ids NOTNULL THEN
        FOR i IN 1 .. array_upper (prm_sec_ids, 1) LOOP
            IF NOT EXISTS (SELECT 1 FROM events.secteur_event WHERE eve_id = prm_eve_id AND sec_id = prm_sec_ids[i])
                INSERT INTO events.secteur_event (eve_id, sec_id) VALUES (prm_eve_id, prm_sec_ids[i]);
            END IF;
        END LOOP;
    END IF;
END;
```



## Name

events\_event\_supprime — Supprime un événement.

## Synopsis

```
void events_event_supprime(prm_token, prm_eve_id);
```

```
int4 prm_token;
```

```
int4 prm_eve_id;
```

## Description

Supprime un événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eve\_id* : Identifiant de l'événement

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  DELETE FROM events.secteur_event WHERE eve_id = prm_eve_id;
  DELETE FROM events.event_personne WHERE eve_id = prm_eve_id;
  DELETE FROM events.event_ressource WHERE eve_id = prm_eve_id;
  DELETE FROM events.event_memo WHERE eve_id = prm_eve_id;
  DELETE FROM events.event WHERE eve_id = prm_eve_id;
END;
```

## Name

events\_event\_type\_ajoute — Ajoute un type d'événement.

## Synopsis

```
int4 events_event_type_ajoute(prm_token, prm_eca_id, prm_intitule);
```

```
int4 prm_token;
```

```
int4 prm_eca_id;
```

```
varchar prm_intitule;
```

## Description

Ajoute un type d'événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eca\_id* : Catégorie d'événement à laquelle appartient le type
- *prm\_intitule* : Intitulé du type

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
DECLARE
    ret integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, TRUE);
    INSERT INTO events.event_type (eca_id, ety_intitule) VALUES (prm_eca_id, prm_intitule) RETURNING ety_id;
    RETURN ret;
END;
```

## Name

`events_event_type_etablissement_get` — Retourne TRUE si un établissement utilise un type d'événement donné, FALSE sinon.

## Synopsis

```
bool events_event_type_etablissement_get(prm_token, prm_ety_id, prm_eta_id);
```

```
int4 prm_token;  
int4 prm_ety_id;  
int4 prm_eta_id;
```

## Description

Retourne TRUE si un établissement utilise un type d'événement donné, FALSE sinon. Entrées :

- `prm_token` : Token d'authentification
- `prm_ety_id` : Type d'événement
- `prm_eta_id` : Identifiant de l'établissement

Remarques : Nécessite les droits à la configuration "Établissement"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  RETURN EXISTS (SELECT 1 FROM events.event_type_etablissement WHERE ety_id = prm_ety_id AND eta_id = prm_eta_id);  
END;
```

## Name

events\_event\_type\_etablissement\_set — Indique si un établissement utilise un type d'événement donné.

## Synopsis

```
void events_event_type_etablissement_set(prm_token, prm_ety_id, prm_eta_id,  
prm_b);
```

```
int4 prm_token;  
int4 prm_ety_id;  
int4 prm_eta_id;  
bool prm_b;
```

## Description

Indique si un établissement utilise un type d'événement donné. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Type d'événement
- *prm\_eta\_id* : Identifiant de l'établissement
- *prm\_b* : TRUE si l'établissement utilise le type, FALSE sinon.

Remarques : Nécessite les droits à la configuration "Établissement"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  IF EXISTS (SELECT 1 FROM events.event_type_etablissement WHERE ety_id = prm_ety_id AND eta_id = prm_eta_id)  
    DELETE FROM events.event_type_etablissement WHERE ety_id = prm_ety_id AND eta_id = prm_eta_id;  
  ELSIF NOT EXISTS (SELECT 1 FROM events.event_type_etablissement WHERE ety_id = prm_ety_id AND eta_id = prm_eta_id)  
    INSERT INTO events.event_type_etablissement (ety_id, eta_id) VALUES (prm_ety_id, prm_eta_id);  
  END IF;  
END;
```

## Name

events\_event\_type\_get — Retourne les informations sur un type d'événement.

## Synopsis

```
event_type events_event_type_get(prm_token, prm_ety_id);
```

```
int4 prm_token;  
int4 prm_ety_id;
```

## Description

Retourne les informations sur un type d'événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Type d'événement

## Source

```
DECLARE  
    ret events.event_type;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM events.event_type WHERE ety_id = prm_ety_id;  
    RETURN ret;  
END;
```

## Name

events\_event\_type\_list — Retourne les types d'événements filtrés par établissement, catégorie et secteurs.

## Synopsis

```
setof      events_event_type_list_all      events_event_type_list(prm_token,
prm_eca_id, prm_sec_ids, prm_eta_id);
```

```
int4 prm_token;
int4 prm_eca_id;
int4[] prm_sec_ids;
int4 prm_eta_id;
```

## Description

Retourne les types d'événements filtrés par établissement, catégorie et secteurs. Entrées :

- prm\_token : Token d'authentification
- prm\_eca\_id : Catégorie d'événements, seuls les types de cette catégorie seront retournés (non NULL)
- prm\_sec\_ids : Secteurs, seuls les types utilisés dans tous ces secteurs seront retournés
- prm\_eta\_id : Identifiant d'établissement, seuls les types utilisés par cet établissement seront retournés, ou NULL

## Source

```
DECLARE
  row events.events_event_type_list_all;
  req varchar;
BEGIN
  PERFORM login_token_assert (prm_token, FALSE, FALSE);
  IF prm_eca_id ISNULL THEN
    RETURN;
  END IF;
  req = 'SELECT DISTINCT eca_nom, event_type.* FROM events.event_type INNER JOIN events.events_categorie '
  IF prm_sec_ids NOTNULL THEN
    FOR i IN 1 .. array_upper (prm_sec_ids, 1) LOOP
      req = req || 'INNER JOIN events.event_type_secteur ets' || i || ' ON ets' || i || '.ety_id = event_'
    END LOOP;
  END IF;
  req = req || 'where eca_id = ' || prm_eca_id;
  IF prm_eta_id NOTNULL THEN
    req = req || ' AND eta_id = ' || prm_eta_id;
  END IF;
  req = req || ' ORDER BY event_type.ety_intitule';
  FOR row IN
    EXECUTE req
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

events\_event\_type\_list\_all — Retourne les types d'événements filtrés par établissement et secteurs, toutes catégories confondues.

## Synopsis

```
setof events_event_type_list_all events_event_type_list_all(prm_token,  
prm_sec_ids, prm_eta_id);
```

```
int4 prm_token;  
int4[] prm_sec_ids;  
int4 prm_eta_id;
```

## Description

Retourne les types d'événements filtrés par établissement et secteurs, toutes catégories confondues. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_sec\_ids* : Secteurs, seuls les types utilisés dans tous ces secteurs seront retournés
- *prm\_eta\_id* : Identifiant d'établissement, seuls les types utilisés par cet établissement seront retournés, ou NULL

## Source

```
DECLARE
    row events_event_type_list_all;
    req varchar;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    req = 'SELECT DISTINCT eca_nom, event_type.ety_id, event_type.eca_id, event_type.ety_intitule COLLATE "C"
        FROM event_type
        WHERE event_type.ety_id = event_type.ety_id';

    IF prm_sec_ids NOTNULL THEN
        FOR i IN 1 .. array_upper (prm_sec_ids, 1) LOOP
            req = req || 'INNER JOIN events.event_type_secteur ets' || i || ' ON ets' || i || '.ety_id = event_type.ety_id';
        END LOOP;
    END IF;

    IF prm_eta_id NOTNULL THEN
        req = req || ' AND eta_id = ' || prm_eta_id;
    END IF;
    req = req || ' ORDER BY event_type.eca_id, event_type.ety_intitule COLLATE "C"';
    FOR row IN
        EXECUTE req
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_event\_type\_list\_par\_evs — Retourne la liste d'événements filtrés par catégories et secteurs de la configuration d'une page d'événements.

## Synopsis

```
setof                                events_event_type_list_par_evs
events_event_type_list_par_evs(prm_token, prm_evs_id);
```

```
int4 prm_token;
int4 prm_evs_id;
```

## Description

Retourne la liste d'événements filtrés par catégories et secteurs de la configuration d'une page d'événements. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identifiant de la configuration de page d'événements

## Source

```
DECLARE
    row events.event_type_list_par_evs;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT event_type.ety_id, ety_intitule FROM events.event_type
        LEFT JOIN events.categorie_events USING(eca_id)
        LEFT JOIN events.event_type_secteur USING(ety_id)
        LEFT JOIN events.secteur_events USING(sec_id)
        WHERE categorie_events.evs_id = prm_evs_id AND secteur_events.evs_id = prm_evs_id
        ORDER BY ety_intitule
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```



## Name

events\_event\_type\_secteur\_ajoute — Affecte un type d'événement à un secteur.

## Synopsis

```
int4 events_event_type_secteur_ajoute(prm_token, prm_ety_id, prm_sec_id);
```

```
int4 prm_token;  
int4 prm_ety_id;  
int4 prm_sec_id;
```

## Description

Affecte un type d'événement à un secteur. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Identifiant du type d'événement
- *prm\_sec\_id* : Identifiant du secteur

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, TRUE);  
    INSERT INTO events.event_type_secteur (ety_id, sec_id) VALUES (prm_ety_id, prm_sec_id) RETURNING ets_id  
    RETURN ret;  
END;
```

## Name

events\_event\_type\_secteur\_list — Retourne la liste des secteurs auxquels est affecté un type d'événement.

## Synopsis

```
setof secteur events_event_type_secteur_list(prm_token, prm_ety_id);
```

```
int4 prm_token;
```

```
int4 prm_ety_id;
```

## Description

Retourne la liste des secteurs auxquels est affecté un type d'événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Identifiant du type d'événement

## Source

```
DECLARE
    row meta.secteur;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT secteur.* FROM meta.secteur
            INNER JOIN events.event_type_secteur USING(sec_id)
            WHERE ety_id = prm_ety_id
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_event\_type\_secteur\_supprime — Supprime l'affectation d'un type d'événement à un secteur.

## Synopsis

```
void events_event_type_secteur_supprime(prm_token, prm_ety_id, prm_sec_id);
```

```
int4 prm_token;  
int4 prm_ety_id;  
int4 prm_sec_id;
```

## Description

Supprime l'affectation d'un type d'événement à un secteur. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Identifiant du type d'événement
- *prm\_sec\_id* : Identifiant du secteur

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, TRUE);  
    DELETE FROM events.event_type_secteur WHERE ety_id = prm_ety_id AND sec_id = prm_sec_id;  
END;
```

## Name

events\_event\_type\_set\_intitule — Modifie l'intitulé d'un type d'événement.

## Synopsis

```
void events_event_type_set_intitule(prm_token, prm_ety_id, prm_intitule);
```

```
int4 prm_token;
```

```
int4 prm_ety_id;
```

```
varchar prm_intitule;
```

## Description

Modifie l'intitulé d'un type d'événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Identifiant du type d'événement
- *prm\_intitule* : Nouvel intitulé

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  UPDATE events.event_type SET ety_intitule = prm_intitule WHERE ety_id = prm_ety_id;
END;
```

## Name

events\_event\_type\_set\_intitule\_individuel — Indique si l'intitulé d'un événement ce ce type peut être personnalisé.

## Synopsis

```
void    events_event_type_set_intitule_individuel(prm_token,    prm_ety_id,  
prm_intitule_individuel);
```

```
int4    prm_token;  
int4    prm_ety_id;  
bool    prm_intitule_individuel;
```

## Description

Indique si l'intitulé d'un événement ce ce type peut être personnalisé. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Identifiant du type d'événement
- *prm\_intitule\_individuel* : TRUE si l'intitulé peut être personnalisé

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, TRUE);  
    UPDATE events.event_type SET ety_intitule_individuel = prm_intitule_individuel WHERE ety_id = prm_ety_id;  
END;
```

## Name

events\_event\_type\_supprime — Supprime un type d'événement.

## Synopsis

```
void events_event_type_supprime(prm_token, prm_ety_id);
```

```
int4 prm_token;  
int4 prm_ety_id;
```

## Description

Supprime un type d'événement. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ety\_id* : Identifiant du type d'événement

Remarques : Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  DELETE FROM events.event_type_secteur WHERE ety_id = prm_ety_id;  
  DELETE FROM events.event_type WHERE ety_id = prm_ety_id;  
END;
```

## Name

events\_events\_add — Crée une nouvelle une vue d'événements.

## Synopsis

```
int4 events_events_add(prm_token, prm_titre, prm_code, prm_ety_id);
```

```
int4 prm_token;  
varchar prm_titre;  
varchar prm_code;  
int4 prm_ety_id;
```

## Description

Crée une nouvelle une vue d'événements. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_titre* : Titre de la nouvelle vue
- *prm\_code* : Code le la nouvelle vue
- *prm\_ety\_id* : Identifiant du type pour filtrer les événements affichées (ou NULL)

Remarques :

- Nécessite les droits à la configuration "Réseau"

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  INSERT INTO events.events (evs_titre, evs_code, ety_id) VALUES (prm_titre, prm_code, prm_ety_id) RETURN  
  RETURN ret;  
END;
```

## Name

events\_events\_categorie\_list — Retourne la liste des catégories d'événements

## Synopsis

```
setof events_categorie events_events_categorie_list(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des catégories d'événements

## Source

```
DECLARE
    row events.events_categorie;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM events.events_categorie ORDER BY eca_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```



## Name

`events_events_copie_et_ajoute_type` — Crée une nouvelle configuration de page d'événements en copiant une configuration existante et en y appliquant un type comme filtre.

## Synopsis

```
int4 events_events_copie_et_ajoute_type(prm_token, prm_evs_id, prm_ety_id);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_ety_id;
```

## Description

Crée une nouvelle configuration de page d'événements en copiant une configuration existante et en y appliquant un type comme filtre. Entrées :

- `prm_token` : Token d'authentification
- `prm_evs_id` : L'identifiant de la configuration de page événement à copier
- `prm_ety_id` : Identifiant du type d'événement à appliquer à la nouvelle configuration

Remarques : Nécessite les droits à la configuration de l'interface

## Source

```
DECLARE
    ret integer;
    id integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    INSERT INTO events.events (evs_titre, evs_code, ety_id)
        SELECT evs_titre || ' ' || ety_intitule, evs_code || '_' || prm_ety_id, prm_ety_id FROM events.events
        WHERE evs_id = prm_evs_id AND event_type.ety_id = prm_ety_id
    RETURNING evs_id INTO ret;
    FOR id IN SELECT DISTINCT eca_id FROM events.categorie_events WHERE evs_id = prm_evs_id LOOP
        INSERT INTO events.categorie_events (eca_id, evs_id) VALUES (id, ret);
    END LOOP;
    FOR id IN SELECT DISTINCT sec_id FROM events.secteur_events WHERE evs_id = prm_evs_id LOOP
        INSERT INTO events.secteur_events (sec_id, evs_id) VALUES (id, ret);
    END LOOP;
    RETURN ret;
END;
```

## Name

events\_events\_get — Retourne les informations d'une configuration de page événement

## Synopsis

```
events events_events_get(prm_token, prm_id);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

## Description

Retourne les informations d'une configuration de page événement

## Source

```
DECLARE
    ret events.events;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM events.events WHERE evs_id = prm_id;
    RETURN ret;
END;
```

## Name

events\_events\_get\_par\_code — Retourne les informations d'une configuration de page événement, par son code

## Synopsis

```
events events_events_get_par_code(prm_token, prm_code);
```

```
int4 prm_token;  
varchar prm_code;
```

## Description

Retourne les informations d'une configuration de page événement, par son code

## Source

```
DECLARE  
    ret events.events;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM events.events WHERE evs_code = prm_code;  
    RETURN ret;  
END;
```

## Name

events\_events\_list — Retourne la liste des configurations de page événement.

## Synopsis

```
setof events events_events_list(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des configurations de page événement. Remarques : Nécessite les droits à la configuration de l'interface

## Source

```
DECLARE
    row events.events;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row IN
        SELECT * FROM events.events ORDER BY evs_titre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_events\_liste\_details

## Synopsis

```
setof events_events_liste_details events_events_liste_details(prm_token);
```

```
int4 prm_token;
```

## Source

```
DECLARE
  row events.events_events_liste_details;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT
      evs_id,
      evs_titre,
      evs_code,
      concatenate (DISTINCT secteur.sec_nom),
      concatenate (DISTINCT events_categorie.eca_nom),
      ety_intitule
    FROM events.events
    LEFT JOIN events.secteur_events USING(evs_id)
    LEFT JOIN meta.secteur USING(sec_id)
    LEFT JOIN events.categorie_events USING(evs_id)
    LEFT JOIN events.events_categorie USING(eca_id)
    LEFT JOIN events.event_type USING(ety_id)
    GROUP BY evs_id, evs_titre, evs_code, ety_intitule
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

events\_events\_supprime — Supprime une configuration de page événement.

## Synopsis

```
void events_events_supprime(prm_token, prm_evs_id);
```

```
int4 prm_token;
```

```
int4 prm_evs_id;
```

## Description

Supprime une configuration de page événement. Remarques : Nécessite les droits à la configuration de l'interface

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM events.categorie_events WHERE evs_id = prm_evs_id;
  DELETE FROM events.secteur_events WHERE evs_id = prm_evs_id;
  DELETE FROM events.events WHERE evs_id = prm_evs_id;
END;
```

## Name

events\_events\_update — Modifie les informations d'une vue d'événements.

## Synopsis

```
void events_events_update(prm_token, prm_evs_id, prm_titre, prm_code,  
prm_ety_id);
```

```
int4 prm_token;  
int4 prm_evs_id;  
varchar prm_titre;  
varchar prm_code;  
int4 prm_ety_id;
```

## Description

Modifie les informations d'une vue d'événements. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identifiant de la vue d'événements
- *prm\_titre* : Nouveau titre de la vue
- *prm\_code* : Nouveau code de la vue
- *prm\_ety\_id* : Identifiant du type pour filtrer les événements affichées (ou NULL)

Remarques :

- Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE events.events SET  
    evs_titre = prm_titre,  
    evs_code = prm_code,  
    ety_id = prm_ety_id  
  WHERE evs_id = prm_evs_id;  
END;
```

## Name

events\_groupe\_liste — Retourne la liste des groupes affectés à un des secteurs de la page d'événement.

## Synopsis

```
setof events_groupe_liste2 events_groupe_liste(prm_token, prm_evs_id);
```

```
int4 prm_token;
```

```
int4 prm_evs_id;
```

## Description

Retourne la liste des groupes affectés à un des secteurs de la page d'événement.

## Source

```
DECLARE
  row events.events_groupe_liste2;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    select DISTINCT groupe.grp_id, groupe.grp_nom, eta_id, eta_nom
      FROM groupe
      INNER JOIN etablisement USING(eta_id)
      INNER JOIN groupe_secteur USING (grp_id)
      INNER JOIN events.secteur_events USING (sec_id)
      where secteur_events.evs_id = prm_evs_id
      ORDER BY eta_nom, grp_nom
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```



## Name

events\_groupe\_liste\_debut — Retourne sous forme d'événements les entrées d'utilisateurs dans des groupes.

## Synopsis

```
setof events_groupe_liste events_groupe_liste_debut(prm_token, prm_evs_id,
prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne sous forme d'événements les entrées d'utilisateurs dans des groupes. Entrées :

- prm\_token : Token d'authentification
- prm\_evs\_id : Identification de la configuration de page d'événement (pour trier sur les groupes du même secteur que la page)
- prm\_per\_id : Identifiant d'un utilisateur (pour spécialiser la recherche à cet utilisateur) ou NULL
- prm\_start : Date de début de période de recherche
- prm\_end : Date de fin de période de recherche
- prm\_grp\_id : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux utilisateurs de ce groupe ou NULL
- prm\_per\_ids : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche à un de ces utilisateurs au moins

## Source

```
DECLARE
    row events.events_groupe_liste;
    p_start timestamp;
    p_end timestamp;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    if prm_start NOTNULL THEN
        p_start = prm_start;
    ELSE
        p_start = timestamp '-INFINITY';
    END IF;
    if prm_end NOTNULL THEN
        p_end = prm_end;
    ELSE
        p_end = timestamp 'INFINITY';
    END IF;
    FOR row IN
        select DISTINCT personne_groupe.per_id,
            personne_info_varchar_get (prm_token, personne_groupe.per_id, 'nom'),
            personne_info_varchar_get (prm_token, personne_groupe.per_id, 'prenom'),
            EXTRACT (EPOCH FROM personne_groupe.peg_debut ),
            grp_nom,
            (SELECT sec_icone FROM meta.secteur INNER JOIN groupe_secteur USING(sec_id) WHERE grp_id = groupe.g
            FROM personne_groupe
            INNER JOIN groupe USING (grp_id)
            INNER JOIN groupe_secteur USING (grp_id)
```

```
--      INNER JOIN meta.secteur USING (sec_id)
      INNER JOIN events.secteur_events USING (sec_id)
      INNER JOIN personne_groupe groupef ON groupef.per_id = personne_groupe.per_id AND (COALESCE (person
      where secteur_events.evs_id = prm_evs_id AND
      personne_groupe.peg_debut between p_start AND p_end AND
      (prm_per_id ISNULL OR personne_groupe.per_id = prm_per_id)
      AND (prm_grp_id ISNULL OR prm_grp_id = groupef.grp_id)
      AND (prm_per_ids ISNULL OR personne_groupe.per_id = ANY(prm_per_ids))
      LOOP
      RETURN NEXT row;
      END LOOP;
END;
```

## Name

events\_groupe\_liste\_fin — Retourne sous forme d'événement les sorties de groupes d'utilisateurs.

## Synopsis

```
setof events_groupe_liste events_groupe_liste_fin(prm_token, prm_evs_id,
prm_per_id, prm_start, prm_end, prm_grp_id, prm_per_ids);
```

```
int4 prm_token;
int4 prm_evs_id;
int4 prm_per_id;
date prm_start;
date prm_end;
int4 prm_grp_id;
int4[] prm_per_ids;
```

## Description

Retourne sous forme d'événement les sorties de groupes d'utilisateurs. Entrées :

- prm\_token : Token d'authentification
- prm\_evs\_id : Identification de la configuration de page d'événement (pour trier sur les groupes du même secteur que la page)
- prm\_per\_id : Identifiant d'un utilisateur (pour spécialiser la recherche à cet utilisateur) ou NULL
- prm\_start : Date de début de période de recherche
- prm\_end : Date de fin de période de recherche
- prm\_grp\_id : Identifiant d'un groupe d'utilisateurs, pour spécialiser la recherche aux utilisateurs de ce groupe ou NULL
- prm\_per\_ids : Tableau d'identifiants d'utilisateurs, pour spécialiser la recherche à un de ces utilisateurs au moins

## Source

```
DECLARE
    row events.events_groupe_liste;
    p_start timestamp;
    p_end timestamp;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    if prm_start NOTNULL THEN
        p_start = prm_start;
    ELSE
        p_start = timestamp '-INFINITY';
    END IF;
    if prm_end NOTNULL THEN
        p_end = prm_end;
    ELSE
        p_end = timestamp 'INFINITY';
    END IF;
    FOR row IN
        select DISTINCT personne_groupe.per_id,
            personne_info_varchar_get (prm_token, personne_groupe.per_id, 'nom'),
            personne_info_varchar_get (prm_token, personne_groupe.per_id, 'prenom'),
            EXTRACT (EPOCH FROM personne_groupe.peg_fin ),
            grp_nom,
            (SELECT sec_icone FROM meta.secteur INNER JOIN groupe_secteur USING(sec_id) WHERE grp_id = groupe.g
            FROM personne_groupe
            INNER JOIN groupe USING (grp_id)
            INNER JOIN groupe_secteur USING (grp_id)
```

```
--      INNER JOIN meta.secteur USING (sec_id)
INNER JOIN events.secteur_events USING (sec_id)
INNER JOIN personne_groupe groupef ON groupef.per_id = personne_groupe.per_id AND (COALESCE (person
where secteur_events.evs_id = prm_evs_id AND
personne_groupe.peg_fin between p_start AND p_end AND
(prm_per_id ISNULL OR personne_groupe.per_id = prm_per_id)
AND (prm_grp_id ISNULL OR prm_grp_id = groupef.grp_id)
AND (prm_per_ids ISNULL OR personne_groupe.per_id = ANY(prm_per_ids))
LOOP
    RETURN NEXT row;
END LOOP;
END;
```

## Name

events\_secteur\_event\_liste — Retourne la liste des secteurs auxquels est affecté un événement.

## Synopsis

```
setof secteur events_secteur_event_liste(prm_token, prm_eve_id);
```

```
int4 prm_token;
```

```
int4 prm_eve_id;
```

## Description

Retourne la liste des secteurs auxquels est affecté un événement.

## Source

```
DECLARE
    row meta.secteur;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT secteur.* FROM meta.secteur
            INNER JOIN events.secteur_event USING (sec_id)
            WHERE eve_id = prm_eve_id
            ORDER BY sec_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

events\_secteur\_events\_liste — Retourne la liste des secteurs sur lesquels une page d'événements est spécialisée, filtrée sur la liste des secteurs pris en charge par un établissement

## Synopsis

```
setof    secteur    events_secteur_events_liste(prm_token,    prm_evs_id,  
prm_eta_id);
```

```
int4    prm_token;  
int4    prm_evs_id;  
int4    prm_eta_id;
```

## Description

Retourne la liste des secteurs sur lesquels une page d'événements est spécialisée, filtrée sur la liste des secteurs pris en charge par un établissement

## Source

```
DECLARE  
    row meta.secteur;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN  
        SELECT secteur.* FROM meta.secteur  
            INNER JOIN events.secteur_events USING (sec_id)  
            INNER JOIN etablisement_secteur USING(sec_id)  
            WHERE evs_id = prm_evs_id AND eta_id = prm_eta_id  
            ORDER BY sec_nom  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

events\_secteur\_events\_set — Indique les secteurs sur lesquels est spécialisée une vue d'événements.

## Synopsis

```
void events_secteur_events_set(prm_token, prm_evs_id, prm_secteurs);
```

```
int4 prm_token;
int4 prm_evs_id;
varchar[] prm_secteurs;
```

## Description

Indique les secteurs sur lesquels est spécialisée une vue d'événements.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_evs\_id* : Identifiant de la vue d'événements
- *prm\_secteurs* : Tableau de codes de secteurs

Remarques :

- Nécessite les droits à la configuration "Réseau"

## Source

```
BEGIN
PERFORM login._token_assert (prm_token, FALSE, TRUE);
DELETE FROM events.secteur_events WHERE evs_id = prm_evs_id;
IF prm_secteurs NOTNULL THEN
  FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP
    INSERT INTO events.secteur_events (evs_id, sec_id) VALUES (prm_evs_id, (SELECT sec_id FROM meta.sec
  END LOOP;
END IF;
END;
```

## 6. liste

### 6.1. Description

Configuration des pages liste. Une page liste est définie par une liste de champs usager, personnel, contact ou famille.

### 6.2. Tables

#### 6.2.1. liste.champ

Les champs constituant une page liste

*cha\_id* (int4)  
Identifiant

*lis\_id* (int4)  
Clé étrangère vers liste.lis\_id

Configuration de liste

inf\_id (int4)

Clé étrangère vers info.inf\_id

Champ personne

cha\_\_groupe\_cycle (bool)

Pour un champ groupe, afficher infos de cycle

cha\_\_groupe\_dernier (bool)

Pour un champ groupe, afficher uniquement dernière appartenance

cha\_libelle (varchar)

Libellé dans la liste

cha\_ordre (int4)

Ordre dans la liste

cha\_filtrer (bool)

Proposer de filtrer sur ce champ

cha\_\_groupe\_contact (bool)

Pour un champ groupe, afficher le contact

cha\_verrouiller (bool)

Si valeurs par défaut, verrouiller ces valeurs

cha\_\_famille\_details (bool)

Pour champ famille, afficher les détails

cha\_champs\_supp (bool)

Pour champ Lien ou Famille, indique si on veut afficher des champs supplémentaires

cha\_\_contact\_filtre\_utilisateur (bool)

Pour un champ Contact, filtre par défaut sur l'utilisateur connecté

## Liens vers cette table

- [default.cha\\_id](#)
- [supp.cha\\_id](#)

### 6.2.2. liste.defaut

Valeurs de filtre d'un champ par défaut

def\_id (int4)

Identifiant

cha\_id (int4)

Clé étrangère vers champ.cha\_id

Champ dans liste auquel est rattachée cette valeur par défaut

def\_valeur\_texte (varchar)

Valeur par défaut pour champ de type texte, etc

def\_valeur\_int (int4)

Valeur par défaut pour champ de type liste de sélection, etc

def\_valeur\_int2 (int4)

Valeur par défaut pour champ de type affectation, etc



def\_ordre (int4)  
Ordre de la valeur par défaut

### 6.2.3. liste.liste

Les configurations de page liste.

lis\_id (int4)  
Identifiant

lis\_nom (varchar)  
Nom de la page

ent\_id (int4)  
Clé étrangère vers entite.ent\_id  
  
Identifiant du type d'entité décrit par la liste (usager, personnel, etc)

lis\_inverse (bool)  
TRUE pour placer les libellés des champs à gauche plutôt qu'en haut du tableau.

lis\_pagination\_tout (bool)  
TRUE pour tout afficher par défaut

### Liens vers cette table

- champ.lis\_id

### 6.2.4. liste.sup

Champs supplémentaires à afficher pour un champ lien ou famille

sup\_id (int4)  
Identifiant

cha\_id (int4)  
Clé étrangère vers champ.cha\_id  
  
Champ dans une liste pour lequel afficher le détail

inf\_id (int4)  
Clé étrangère vers info.inf\_id  
  
Champ à afficher en détail

sup\_ordre (int4)  
Ordre d'affichage

## 6.3. Types

### 6.3.1. liste.liste\_champ\_liste\_details

## 6.4. Fonctions

## Name

`jours_avant_anniversaire` — Retourne le nombre de jours avant le prochain anniversaire de la date donnée.

## Synopsis

```
int4 jours_avant_anniversaire(prm_token, prm_date);
```

```
int4 prm_token;
```

```
date prm_date;
```

## Description

Retourne le nombre de jours avant le prochain anniversaire de la date donnée.

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT extract (doy FROM prm_date) - extract (doy FROM CURRENT_TIMESTAMP) INTO ret;
  IF ret < 0 THEN
    ret = ret + 365;
  END IF;
  RETURN ret;
END;
```

## Name

liste\_champ\_add — Ajoute un champ à une configuration de page liste.

## Synopsis

```
int4 liste_champ_add(prm_token, prm_lis_id, prm_inf_id, prm_ordre);
```

```
int4 prm_token;  
int4 prm_lis_id;  
int4 prm_inf_id;  
int4 prm_ordre;
```

## Description

Ajoute un champ à une configuration de page liste.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO liste.champ (lis_id, inf_id, cha_ordre) VALUES (prm_lis_id, prm_inf_id, prm_ordre) RETURNING  
    RETURN ret;  
END;
```

## Name

liste\_champ\_get — Retourne la configuration d'un champ dans une page liste

## Synopsis

```
champ liste_champ_get(prm_token, prm_cha_id);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

## Description

Retourne la configuration d'un champ dans une page liste

## Source

```
DECLARE
  ret liste.champ;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM liste.champ WHERE cha_id = prm_cha_id;
  RETURN ret;
END;
```

## Name

liste\_champ\_liste — Retourne la liste des configurations de page liste

## Synopsis

```
setof champ liste_champ_liste(prm_token, prm_lis_id);
```

```
int4 prm_token;  
int4 prm_lis_id;
```

## Description

Retourne la liste des configurations de page liste

## Source

```
DECLARE  
  row liste.champ;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT * FROM liste.champ WHERE lis_id = prm_lis_id ORDER BY cha_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

liste\_champ\_liste\_details — Retourne la liste détaillée des configurations de page liste.

## Synopsis

```
setof liste_champ_liste_details liste_champ_liste_details(prm_token,  
prm_lis_id);
```

```
int4 prm_token;  
int4 prm_lis_id;
```

## Description

Retourne la liste détaillée des configurations de page liste.

## Source

```

DECLARE
    row liste.liste_champ_liste_details;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT
            champ.cha_id,
            champ.cha_groupe_contact,
            champ.cha_groupe_cycle ,
            champ.cha_groupe_dernier ,
            cha_libelle ,
            cha_ordre ,
            cha_filtrer,
            cha_verrouiller,
            cha_famille_details,
            cha_champs_supp,
            info.inf_libelle,
            infos_type.int_code,
            infos_type.int_libelle,
            info.inf_multiple,
            cha_contact_filtre_utilisateur
        FROM liste.champ
        INNER JOIN meta.info USING(inf_id)
        INNER JOIN meta.infos_type USING (int_id)
        WHERE lis_id = prm_lis_id
        ORDER BY cha_ordre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

liste\_champ\_set\_champs\_supp — Indique si des champs supplémentaires seront affichés dans ce champs.

## Synopsis

```
void liste_champ_set_champs_supp(prm_token, prm_cha_id, prm_champs_supp);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

```
bool prm_champs_supp;
```

## Description

Indique si des champs supplémentaires seront affichés dans ce champs.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE liste.champ SET cha_champs_supp = prm_champs_supp WHERE cha_id = prm_cha_id;
END;
```

## Name

liste\_champ\_set\_contact — Pour un champ groupe, indique si le contact doit être affiché.

## Synopsis

```
void liste_champ_set_contact(prm_token, prm_cha_id, prm_contact);
```

```
int4 prm_token;  
int4 prm_cha_id;  
bool prm_contact;
```

## Description

Pour un champ groupe, indique si le contact doit être affiché.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.champ SET cha_groupe_contact = prm_contact WHERE cha_id = prm_cha_id;  
END;
```



## Name

`liste_champ_set_contact_filtre_utilisateur` — Indique si un champ Contact filtré le sera par défaut sur l'utilisateur connecté.

## Synopsis

```
void liste_champ_set_contact_filtre_utilisateur(prm_token, prm_cha_id,  
prm_filtre_utilisateur);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

```
bool prm_filtre_utilisateur;
```

## Description

Indique si un champ Contact filtré le sera par défaut sur l'utilisateur connecté.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.champ SET cha__contact_filtre_utilisateur = prm_filtre_utilisateur WHERE cha_id = prm_cha_id;  
END;
```

## Name

liste\_champ\_set\_cycle — Pour un champ groupe, indique si les informations de cycle doivent être affichées.

## Synopsis

```
void liste_champ_set_cycle(prm_token, prm_cha_id, prm_cycle);
```

```
int4 prm_token;  
int4 prm_cha_id;  
bool prm_cycle;
```

## Description

Pour un champ groupe, indique si les informations de cycle doivent être affichées.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.champ SET cha_groupe_cycle = prm_cycle WHERE cha_id = prm_cha_id;  
END;
```

## Name

`liste_champ_set_dernier` — Pour un champ groupe, indique si uniquement la dernière appartenance doit être affichée.

## Synopsis

```
void liste_champ_set_dernier(prm_token, prm_cha_id, prm_dernier);
```

```
int4 prm_token;  
int4 prm_cha_id;  
bool prm_dernier;
```

## Description

Pour un champ groupe, indique si uniquement la dernière appartenance doit être affichée.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.champ SET cha_groupe_dernier = prm_dernier WHERE cha_id = prm_cha_id;  
END;
```

## Name

liste\_champ\_set\_details — Pour un champ famille, indique si les détails du lien familial doivent être affichés.

## Synopsis

```
void liste_champ_set_details(prm_token, prm_cha_id, prm_details);
```

```
int4 prm_token;  
int4 prm_cha_id;  
bool prm_details;
```

## Description

Pour un champ famille, indique si les détails du lien familial doivent être affichés.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.champ SET cha_famille_details = prm_details WHERE cha_id = prm_cha_id;  
END;
```

## Name

liste\_champ\_set\_filtreur — Indique s'il est possible de filtrer sur ce champ.

## Synopsis

```
void liste_champ_set_filtreur(prm_token, prm_cha_id, prm_filtreur);
```

```
int4 prm_token;  
int4 prm_cha_id;  
bool prm_filtreur;
```

## Description

Indique s'il est possible de filtrer sur ce champ.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.champ SET cha_filtreur = prm_filtreur WHERE cha_id = prm_cha_id;  
  IF NOT prm_filtreur THEN  
    UPDATE liste.champ SET cha_verrouiller = FALSE WHERE cha_id = prm_cha_id;  
  END IF;  
  DELETE FROM liste.default WHERE cha_id = prm_cha_id;  
END;
```

## Name

liste\_champ\_set\_libelle — Modifie le libellé du champ dans la liste.

## Synopsis

```
void liste_champ_set_libelle(prm_token, prm_cha_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

```
varchar prm_libelle;
```

## Description

Modifie le libellé du champ dans la liste.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE liste.champ SET cha_libelle = prm_libelle WHERE cha_id = prm_cha_id;
END;
```

## Name

liste\_champ\_set\_ordre — Modifie l'ordre du champ dans la liste.

## Synopsis

```
void liste_champ_set_ordre(prm_token, prm_cha_id, prm_ordre);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

```
int4 prm_ordre;
```

## Description

Modifie l'ordre du champ dans la liste.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE liste.champ SET cha_ordre = prm_ordre WHERE cha_id = prm_cha_id;
END;
```

## Name

liste\_champ\_set\_verrouiller — Indique si les valeurs de filtrage par défaut sont verrouillées.

## Synopsis

```
void liste_champ_set_verrouiller(prm_token, prm_cha_id, prm_verrouiller);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

```
bool prm_verrouiller;
```

## Description

Indique si les valeurs de filtrage par défaut sont verrouillées.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE liste.champ SET cha_verrouiller = prm_verrouiller WHERE cha_id = prm_cha_id;
END;
```



## Name

liste\_champ\_supprime — Supprime un champ d'une page liste.

## Synopsis

```
void liste_champ_supprime(prm_token, prm_cha_id);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

## Description

Supprime un champ d'une page liste.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM liste.champ WHERE cha_id = prm_cha_id;
END;
```

## Name

liste\_defaut\_ajoute\_groupe — Ajoute une valeur par défaut à un champ de type affectation, etc

## Synopsis

```
int4 liste_defaut_ajoute_groupe(prm_token, prm_cha_id, prm_val, prm_val2);
```

```
int4 prm_token;  
int4 prm_cha_id;  
int4 prm_val;  
int4 prm_val2;
```

## Description

Ajoute une valeur par défaut à un champ de type affectation, etc

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  INSERT INTO liste.defaut (cha_id, def_valeur_int, def_valeur_int2) VALUES (prm_cha_id, prm_val, prm_val2);  
  RETURNING def_id INTO ret;  
  RETURN ret;  
END;
```

## Name

liste\_defaut\_ajoute\_selection — Ajoute une valeur par défaut à un champ de type sélection, etc

## Synopsis

```
int4 liste_defaut_ajoute_selection(prm_token, prm_cha_id, prm_val);
```

```
int4 prm_token;  
int4 prm_cha_id;  
int4 prm_val;
```

## Description

Ajoute une valeur par défaut à un champ de type sélection, etc

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO liste.defaut (cha_id, def_valeur_int) VALUES (prm_cha_id, prm_val)  
        RETURNING def_id INTO ret;  
    RETURN ret;  
END;
```

## Name

liste\_defaut\_ajoute\_texte — Ajoute une valeur par défaut à un champ de type texte, etc

## Synopsis

```
int4 liste_defaut_ajoute_texte(prm_token, prm_cha_id, prm_val);
```

```
int4 prm_token;  
int4 prm_cha_id;  
varchar prm_val;
```

## Description

Ajoute une valeur par défaut à un champ de type texte, etc

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  INSERT INTO liste.defaut (cha_id, def_valeur_texte) VALUES (prm_cha_id, prm_val)  
    RETURNING def_id INTO ret;  
  RETURN ret;  
END;
```

## Name

liste\_defaut\_liste — Retourne la liste des valeurs de filtrage par défaut d'un champ dans la liste.

## Synopsis

```
setof default liste_defaut_liste(prm_token, prm_cha_id);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

## Description

Retourne la liste des valeurs de filtrage par défaut d'un champ dans la liste.

## Source

```
DECLARE
    row liste.default;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM liste.default WHERE cha_id = prm_cha_id ORDER BY def_ordre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

liste\_defaut\_supprime — Supprime une valeur de filtrage par défaut pour un champ dans la liste.

## Synopsis

```
void liste_defaut_supprime(prm_token, prm_def_id);
```

```
int4 prm_token;
```

```
int4 prm_def_id;
```

## Description

Supprime une valeur de filtrage par défaut pour un champ dans la liste.

## Source

```
DECLARE
  cha integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT cha_id INTO cha FROM liste.defaut WHERE def_id = prm_def_id;
  DELETE FROM liste.defaut WHERE def_id = prm_def_id;
  IF NOT EXISTS (SELECT 1 FROM liste.defaut WHERE cha_id = cha) THEN
    UPDATE liste.champ SET cha_verrouiller = FALSE WHERE cha_id = cha;
  END IF;
END;
```

## Name

liste\_liste\_all — Retourne la liste des configurations de pages liste.

## Synopsis

```
setof liste liste_liste_all(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des configurations de pages liste.

## Source

```
DECLARE
    row liste.liste;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row IN
        SELECT * FROM liste.liste
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

liste\_liste\_create — Crée une nouvelle configuration de page liste.

## Synopsis

```
int4  liste_liste_create(prm_token,  prm_nom,  prm_ent_id,  prm_inverse,  
prm_pagination_tout);
```

```
int4 prm_token;  
varchar prm_nom;  
int4 prm_ent_id;  
bool prm_inverse;  
bool prm_pagination_tout;
```

## Description

Crée une nouvelle configuration de page liste.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO liste.liste (lis_nom, ent_id, lis_inverse, lis_pagination_tout) VALUES (prm_nom, prm_ent_id,  
    RETURN ret;  
END;
```



## Name

liste\_liste\_get — Retourne les informations d'une configuration de page liste.

## Synopsis

```
liste liste_liste_get(prm_token, prm_lis_id);
```

```
int4 prm_token;
```

```
int4 prm_lis_id;
```

## Description

Retourne les informations d'une configuration de page liste.

## Source

```
DECLARE
    ret liste.liste;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM liste.liste WHERE lis_id = prm_lis_id;
    RETURN ret;
END;
```

## Name

liste\_liste\_supprime — Supprime une configuration de page liste.

## Synopsis

```
void liste_liste_supprime(prm_token, prm_lis_id);
```

```
int4 prm_token;  
int4 prm_lis_id;
```

## Description

Supprime une configuration de page liste.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.topsousmenu SET tsm_type_id = NULL WHERE tsm_type = 'liste' AND tsm_type_id = prm_lis_id;  
  DELETE FROM liste.default WHERE cha_id IN (SELECT cha_id FROM liste.champ WHERE lis_id = prm_lis_id);  
  DELETE FROM liste.champ WHERE lis_id = prm_lis_id;  
  DELETE FROM liste.liste WHERE lis_id = prm_lis_id;  
END;
```

## Name

liste\_liste\_update — Modifie les informations d'une configuration de page liste.

## Synopsis

```
void  liste_liste_update(prm_token,  prm_lis_id,  prm_nom,  prm_ent_id,  
prm_inverse, prm_pagination_tout);
```

```
int4  prm_token;  
int4  prm_lis_id;  
varchar prm_nom;  
int4  prm_ent_id;  
bool  prm_inverse;  
bool  prm_pagination_tout;
```

## Description

Modifie les informations d'une configuration de page liste.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE liste.liste SET lis_nom = prm_nom, ent_id = prm_ent_id, lis_inverse = prm_inverse, lis_pagination_tout = prm_pagination_tout;  
END;
```

## Name

liste\_supp\_edit — Indique la liste des champs à afficher pour détailler une colonne de tableau donnée.

## Synopsis

```
void liste_supp_edit(prm_token, prm_cha_id, prm_inf_ids);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

```
int4[] prm_inf_ids;
```

## Description

Indique la liste des champs à afficher pour détailler une colonne de tableau donnée.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM liste.supp WHERE cha_id = prm_cha_id;
  FOR i IN 1 .. array_upper(prm_inf_ids, 1) LOOP
    INSERT INTO liste.supp (cha_id, inf_id, sup_ordre)
      VALUES (prm_cha_id, prm_inf_ids[i], i-1);
  END LOOP;
END;
```

## Name

liste\_supp\_list — Retourne la liste des champs afficher pour détailler une colonne de tableau.

## Synopsis

```
setof info liste_supp_list(prm_token, prm_cha_id);
```

```
int4 prm_token;
```

```
int4 prm_cha_id;
```

## Description

Retourne la liste des champs afficher pour détailler une colonne de tableau.

## Source

```
DECLARE
  row meta.info;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT info.* FROM liste.supp
      INNER JOIN meta.info USING(inf_id)
      WHERE (prm_cha_id ISNULL OR cha_id = prm_cha_id)
      ORDER BY sup_ordre
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

liste\_supp\_supprime — Supprime un champs supplémentaire.

## Synopsis

```
void liste_supp_supprime(prm_token, prm_cha_id, prm_inf_id);
```

```
int4 prm_token;  
int4 prm_cha_id;  
int4 prm_inf_id;
```

## Description

Supprime un champs supplémentaire.

## Source

```
BEGIN  
DELETE FROM liste.supp WHERE cha_id = prm_cha_id AND inf_id = prm_inf_id;  
END;
```

# 7. localise

## 7.1. Description

Système de localisation. A ce point, il est possible de localiser un terme pour un secteur particulier.

## 7.2. Tables

### 7.2.1. localise.localisation\_secteur

Localisation d'un terme pour un secteur particulier.

loc\_id (int4)

ter\_id (int4)

Clé étrangère vers terme.ter\_id

loc\_valeur (varchar)

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

### 7.2.2. localise.termes

Terme à localiser.

ter\_id (int4)

ter\_code (varchar)

ter\_commentaire (varchar)

## Liens vers cette table

- localisation\_secteur.ter\_id

## **7.3. Types**

### **7.3.1. localise.localise\_terme\_liste\_details**

## **7.4. Fonctions**

## Name

localise\_par\_code\_secteur — Retourne un terme localisé pour un secteur donné.

## Synopsis

```
vvarchar localise_par_code_secteur(prm_token, prm_ter_code, prm_sec_code);
```

```
int4 prm_token;
```

```
vvarchar prm_ter_code;
```

```
vvarchar prm_sec_code;
```

## Description

Retourne un terme localisé pour un secteur donné.

## Source

```
DECLARE
    ret varchar;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    select loc_valeur INTO ret from localise.localisation_secteur
    inner join localise.terme using(ter_id)
    left join meta.secteur using(sec_id)
    WHERE ter_code = prm_ter_code AND (sec_code = prm_sec_code OR sec_code ISNULL)
    ORDER BY sec_code LIMIT 1;
    RETURN ret;
END;
```



## Name

localise\_par\_code\_secteur\_set — Modifie la localisation d'un terme pour un secteur particulier.

## Synopsis

```
void localise_par_code_secteur_set(prm_token, prm_ter_code, prm_sec_code,  
prm_valeur);
```

```
int4 prm_token;  
varchar prm_ter_code;  
varchar prm_sec_code;  
varchar prm_valeur;
```

## Description

Modifie la localisation d'un terme pour un secteur particulier. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_ter\_code* : Code du terme
- *prm\_sec\_code* : Code du secteur ou NULL pour modifier la valeur par défaut
- *prm\_valeur* : Valeur de la localisation

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE localise.localisation_secteur  
    SET loc_valeur = prm_valeur  
    WHERE  
      ter_id = (SELECT ter_id FROM localise.termes WHERE ter_code = prm_ter_code) AND  
      ( (prm_sec_code ISNULL AND sec_id ISNULL) OR  
        sec_id = (SELECT sec_id FROM meta.secteur WHERE sec_code = prm_sec_code) );  
  IF NOT FOUND THEN  
    INSERT INTO localise.localisation_secteur (ter_id, loc_valeur, sec_id)  
      VALUES ((SELECT ter_id FROM localise.termes WHERE ter_code = prm_ter_code),  
              prm_valeur,  
              (SELECT sec_id FROM meta.secteur WHERE sec_code = prm_sec_code) );  
  END IF;  
END;
```

## Name

localise\_par\_code\_secteur\_supprime — Supprime la localisation d'un terme pour un secteur donné.

## Synopsis

```
void      localise_par_code_secteur_supprime(prm_token,      prm_ter_code,  
prm_sec_code);
```

```
int4 prm_token;  
varchar prm_ter_code;  
varchar prm_sec_code;
```

## Description

Supprime la localisation d'un terme pour un secteur donné.

## Source

```
BEGIN  
  DELETE FROM localise.localisation_secteur  
  WHERE  
    sec_id = (SELECT sec_id FROM meta.secteur WHERE sec_code = prm_sec_code) AND  
    ter_id = (SELECT ter_id FROM localise.termes WHERE ter_code = prm_ter_code);  
END;
```

## Name

`localise_terme_get` — Retourne les détails d'un terme à localiser.

## Synopsis

```
terme localise_terme_get(prm_token, prm_id);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

## Description

Retourne les détails d'un terme à localiser.

## Source

```
DECLARE
  ret localise.terme;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  SELECT * INTO ret FROM localise.terme WHERE ter_id = prm_id;
  RETURN ret;
END;
```

## Name

localise\_terme\_liste\_details — Retourne le détail des termes à localiser.

## Synopsis

```
setof localise_terme_liste_details localise_terme_liste_details(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne le détail des termes à localiser.

## Source

```
DECLARE
  row localise.localise_terme_liste_details;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  FOR row IN
    SELECT ter_id, ter_code, ter_commentaire, localise.localise_par_code_secteur(prm_token, ter_code, N
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

localise\_terme\_set — Modifie les détails d'un terme à localiser.

## Synopsis

```
void localise_terme_set(prm_token, prm_id, prm_commentaire);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

```
varchar prm_commentaire;
```

## Description

Modifie les détails d'un terme à localiser.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  UPDATE localise.terme SET ter_commentaire = prm_commentaire WHERE ter_id = prm_id;
END;
```

## Name

localise\_terme\_supprime — Supprime un terme à localiser.

## Synopsis

```
void localise_terme_supprime(prm_token, prm_ter_id);
```

```
int4 prm_token;
```

```
int4 prm_ter_id;
```

## Description

Supprime un terme à localiser.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  DELETE FROM localise.localisation_secteur WHERE ter_id = prm_ter_id;
  DELETE FROM localise.terme WHERE ter_id = prm_ter_id;
END;
```

## 8. lock

### 8.1. Description

Système de verrouillage de fiches.

### 8.2. Tables

#### 8.2.1. lock.fiche

Fiche verrouillée.

*loc\_id* (int4)

Identifiant

*uti\_id* (int4)

Clé étrangère vers utilisateur.*uti\_id*

Utilisateur ayant verrouillé la fiche

*per\_id* (int4)

Clé étrangère vers personne.*per\_id*

Identifiant de la personne affichée par la fiche

*loc\_date* (timestampz)

Date de dernière consultation de la fiche par l'utilisateur

*sme\_id* (int4)

Dernier menu visité dans la fiche

### 8.3. Fonctions

## Name

fiche\_lock — Verouille une fiche.

## Synopsis

```
int4 fiche_lock(prm_token, prm_per_id, prm_force);
```

```
int4 prm_token;  
int4 prm_per_id;  
bool prm_force;
```

## Description

Verouille une fiche.

## Source

```
DECLARE  
    uti integer;  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;  
    DELETE FROM lock.fiche WHERE AGE(CURRENT_TIMESTAMP, loc_date) > '5 min';  
    IF NOT prm_force THEN  
        IF EXISTS (SELECT 1 FROM lock.fiche WHERE per_id = prm_per_id AND uti_id <> uti) THEN  
            SELECT uti_id INTO ret FROM lock.fiche WHERE per_id = prm_per_id AND uti_id <> uti LIMIT 1;  
            RETURN -ret;  
        ELSIF EXISTS (SELECT 1 FROM lock.fiche WHERE per_id = prm_per_id AND uti_id = uti) THEN  
            DELETE FROM lock.fiche WHERE per_id = prm_per_id AND uti_id = uti;  
        END IF;  
  
        INSERT INTO lock.fiche (uti_id, per_id, loc_date) VALUES (uti, prm_per_id, CURRENT_TIMESTAMP)  
        RETURNING loc_id INTO ret;  
        RETURN ret;  
    ELSE  
        INSERT INTO lock.fiche (uti_id, per_id, loc_date) VALUES (uti, prm_per_id, CURRENT_TIMESTAMP)  
        RETURNING loc_id INTO ret;  
        RETURN ret;  
  
    END IF;  
END;
```

## Name

fiche\_touch — Rafraîchit la date de dernière consultation d'une fiche.

## Synopsis

```
void fiche_touch(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Description

Rafraîchit la date de dernière consultation d'une fiche.

## Source

```
DECLARE
    uti integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    UPDATE lock.fiche SET loc_date = CURRENT_TIMESTAMP WHERE uti_id = uti AND per_id = prm_per_id;
END;
```



## Name

fiche\_unlock — Déverrouille une fiche.

## Synopsis

```
void fiche_unlock(prm_token, prm_per_id);
```

```
int4 prm_token;  
int4 prm_per_id;
```

## Description

Déverrouille une fiche.

## Source

```
DECLARE  
    uti integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;  
    DELETE FROM lock.fiche WHERE uti_id = uti AND per_id = prm_per_id;  
END;
```

## Name

fiche\_unlock\_tout — Déverrouille toutes les fichiers verrouillées par l'utilisateur authentifié.

## Synopsis

```
void fiche_unlock_tout(prm_token);
```

```
int4 prm_token;
```

## Description

Déverrouille toutes les fichiers verrouillées par l'utilisateur authentifié.

## Source

```
DECLARE
    uti integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    DELETE FROM lock.fiche WHERE uti_id = uti;
END;
```

## Name

lock\_fiche\_liste — Retourne la liste des fiches verrouillées par l'utilisateur authentifié.

## Synopsis

```
setof fiche lock_fiche_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des fiches verrouillées par l'utilisateur authentifié.

## Source

```
DECLARE
  uti integer;
  row lock.fiche;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
  DELETE FROM lock.fiche WHERE uti_id = uti AND AGE(CURRENT_TIMESTAMP, loc_date) > '5 min';
  FOR row IN
    SELECT * FROM lock.fiche WHERE uti_id = uti ORDER BY loc_date DESC
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

lock\_fiche\_set\_sme — Modifie le dernier menu visité.

## Synopsis

```
void lock_fiche_set_sme(prm_per_id, prm_token, prm_sme_id);
```

```
int4 prm_per_id;
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

## Description

Modifie le dernier menu visité.

## Source

```

DECLARE
    uti integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    UPDATE lock.fiche SET sme_id = prm_sme_id WHERE per_id = prm_per_id AND uti_id = uti;
END;
```

# 9. login

## 9.1. Description

Procédures d'authentification de l'utilisateur. Gestion des utilisateurs et groupes d'utilisateurs. L'utilisateur s'authentifie auprès du webservice avec son login et mot de passe à l'aide de la fonction utilisateur.utilisateur\_login2 (prm\_login, prm\_mdp) Cette fonction retourne :

- son identifiant
- un token d'authentification
- les droits de l'utilisateur à configurer le réseau et/ou l'établissement
- la liste des paires (etablissement/portail) auxquels l'utilisateur a droit de se connecter
- si son mot de passe est provisoire

Le token d'authentification, qui devra être utilisé dans toute fonction à accès restreint, permet d'authentifier de nouveau l'utilisateur sans avoir à envoyer de nouveau le login et mot de passe. Le token est valable durant un laps de temps configurable, et au plus tard jusqu'à la déconnexion de l'utilisateur.

Les droits de l'utilisateur à configurer le réseau et/ou l'établissement sont donnés pour information. L'interface graphique pourra par exemple, selon ces droits, afficher ou non les menus correspondants. Les fonctions de configuration du réseau et de l'établissement ont un accès restreint (un token devra être donné). Elles retourneront une erreur en cas d'accès illicite, et une alerte de sécurité pourra être levée.

Un certain nombre de fonctions demandent un établissement et/ou portail en paramètre. Ces paramètres devront être renseignés avec des établissement/portail auxquels l'utilisateur a accès. Elles retourneront une erreur en cas d'accès illicite à un établissement/portail. Si ces paramètres ne sont pas renseignés lors de l'appel à la fonction, celle-ci agira sur tous les établissements/portails auxquels à accès l'utilisateur.

Un mot de passe provisoire est stocké en clair dans la base de données. Un utilisateur ayant droit de configuration de l'établissement a accès en lecture à ces mots de passe provisoires, et a le droit d'écraser un mot de passe non

provisoire avec un nouveau mot de passe provisoire. Un utilisateur peut à tout moment définir un nouveau mot de passe pour son compte, qui devient alors non provisoire.

## 9.2. Tables

### 9.2.1. login.grouputil

Liste des groupes d'utilisateurs.

gut\_id (int4)

Identifiant du groupe d'utilisateurs

gut\_nom (varchar)

Nom du groupe

#### Liens vers cette table

- grouputil\_groupe.gut\_id
- grouputil\_portail.gut\_id
- utilisateur\_grouputil.gut\_id

### 9.2.2. login.grouputil\_groupe

Groupes d'usagers auxquels un groupe d'utilisateurs a accès

ggr\_id (int4)

Identifiant de la relation

gut\_id (int4)

Clé étrangère vers grouputil.gut\_id

Identifiant du groupe d'utilisateurs

grp\_id (int4)

Clé étrangère vers groupe.grp\_id

Identifiant du groupe d'usagers

### 9.2.3. login.grouputil\_portail

Portails auxquels un groupe d'utilisateurs a accès

gpo\_id (int4)

Identifiant de la relation

gut\_id (int4)

Clé étrangère vers grouputil.gut\_id

Identifiant du groupe d'utilisateurs

por\_id (int4)

Clé étrangère vers portail.por\_id

Identifiant du portail

### 9.2.4. login.token

Token d'authentification (interne)

tok\_id (int4)

Identifiant

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Utilisateur ayant reçu ce token

tok\_token (int4)

Valeur du token

tok\_date\_creation (timestampz)

Date de création (à la connexion de l'utilisateur)

### 9.2.5. login.utilisateur

Utilisateurs de l'application

uti\_id (int4)

Identifiant de l'utilisateur

uti\_login (varchar)

Login de connexion

uti\_salt (varchar)

Mot de passe crypté

uti\_root (bool)

Droit de configuration "Réseau"

uti\_config (bool)

Droit de configuration "Etablissement"

per\_id (int4)

Clé étrangère vers personne.per\_id

Identifiant du personnel associé à l'utilisateur

uti\_pwd (varchar)

Mot de passe temporaire en clair

uti\_digest (varchar)

Mot de passe crypté pour connexion par webdav

### Liens vers cette table

- document.uti\_id\_creation
- event.uti\_id\_creation
- fiche.uti\_id
- token.uti\_id
- utilisateur\_grouputil.uti\_id
- note.uti\_id\_auteur
- note\_destinataire.uti\_id
- personne\_info\_boolean.uti\_id

- `personne_info_date.uti_id`
- `personne_info_integer.uti_id`
- `personne_info_integer2.uti_id`
- `personne_info_text.uti_id`
- `personne_info_varchar.uti_id`

### **9.2.6. login.utilisateur\_grouputil**

Affectation des utilisateurs aux groupes d'utilisateurs.

`ugr_id` (int4)

Identifiant de la relation

`uti_id` (int4)

Clé étrangère vers `utilisateur.uti_id`

Identifiant de l'utilisateur

`gut_id` (int4)

Clé étrangère vers `grouputil.gut_id`

Identifiant du groupe d'utilisateurs

## **9.3. Types**

### **9.3.1. login.utilisateur\_login2**

### **9.3.2. login.utilisateur\_liste\_details\_configuration**

### **9.3.3. login.utilisateur\_usagers\_liste**

## **9.4. Fonctions**

## Name

`_token_assert` — [INTERNE] Vérifie la validité d'un token et ses droits de configuration "Établissement" et "Réseau".

## Synopsis

```
void _token_assert(prm_token, prm_config, prm_root);
```

```
int4  prm_token;
bool  prm_config;
bool  prm_root;
```

## Description

[INTERNE] Vérifie la validité d'un token et ses droits de configuration "Établissement" et "Réseau".

Entrée :

- `prm_token` : Token à vérifier
- `prm_config` : Vérifie le droit de configuration "Établissement"
- `prm_root` : Vérifie le droit de configuration "Réseau"

Lève une exception `"insufficient_privilege"` si le token est invalide ou si une demande de vérification config ou root échoue.

## Source

```
DECLARE
    uti integer;
    util login.utilisateur;
BEGIN
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    IF NOT FOUND THEN
        RAISE EXCEPTION USING ERRCODE = 'insufficient_privilege';
    END IF;
    IF uti ISNULL THEN -- C'est le token de l'éditeur
        RETURN;
    END IF;
    IF prm_config AND prm_root THEN
        SELECT * INTO util FROM login.utilisateur WHERE uti_id = uti;
        IF util.uti_config IS DISTINCT FROM TRUE OR util.uti_root IS DISTINCT FROM TRUE THEN
            RAISE EXCEPTION USING ERRCODE = 'insufficient_privilege';
        END IF;
    ELSIF prm_config THEN
        SELECT * INTO util FROM login.utilisateur WHERE uti_id = uti;
        IF util.uti_config IS DISTINCT FROM TRUE THEN
            RAISE EXCEPTION USING ERRCODE = 'insufficient_privilege';
        END IF;
    ELSIF prm_root THEN
        SELECT * INTO util FROM login.utilisateur WHERE uti_id = uti;
        IF util.uti_root IS DISTINCT FROM TRUE THEN
            RAISE EXCEPTION USING ERRCODE = 'insufficient_privilege';
        END IF;
    END IF;
END;
```



## Name

`_token_assert_interface` — [INTERNE] Vérifie les droits de configuration de l'interface pour un token

## Synopsis

```
void _token_assert_interface(prm_token);
```

```
int4 prm_token;
```

## Description

[INTERNE] Vérifie les droits de configuration de l'interface pour un token

## Source

```
DECLARE
  uti integer;
  util login.utilisateur;
BEGIN
  -- TODO : Vérifier les droits de config de l'interface
END;
```

## Name

`_token_create` — [INTERNE] Crée un token d'authentification pour un utilisateur donné.

## Synopsis

```
vvarchar _token_create(prm_uti_id);
```

```
int4 prm_uti_id;
```

## Description

[INTERNE] Crée un token d'authentification pour un utilisateur donné. Retourne la valeur du token, à utiliser dans tout appel de fonction sécurisée

## Source

```
DECLARE
    tok integer DEFAULT NULL;
    trouve BOOLEAN DEFAULT true;
BEGIN
    DELETE FROM login.token WHERE uti_id = prm_uti_id AND AGE(CURRENT_TIMESTAMP, tok_date_creation) > '8 ho
    SELECT tok_token INTO tok FROM login.token WHERE uti_id = prm_uti_id;
        IF tok NOTNULL THEN
            RETURN tok;
        ELSE
            WHILE trouve LOOP
                tok = (RANDOM()*1000000000)::int;
                IF NOT EXISTS (SELECT 1 FROM login.token WHERE tok_token = tok) THEN
                    trouve = false;
                END IF;
            END LOOP;
            INSERT INTO login.token (uti_id, tok_token, tok_date_creation)
                VALUES (prm_uti_id, tok, CURRENT_TIMESTAMP);
            RETURN tok;
        END IF;
END;
```

## Name

login\_grouputil\_add — Ajoute un nouveau groupe d'utilisateurs.

## Synopsis

```
int4 login_grouputil_add(prm_token, prm_nom);
```

```
int4 prm_token;  
varchar prm_nom;
```

## Description

Ajoute un nouveau groupe d'utilisateurs.

Entrée :

- *prm\_token* : token d'authentification
- *prm\_nom* : Nom du groupe

Retour :

- Id du groupe d'utilisateurs créé

Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, TRUE, FALSE);  
    INSERT INTO login.grouputil (gut_nom) VALUES (prm_nom) RETURNING gut_id INTO ret;  
    RETURN ret;  
END;
```

## Name

login\_grouputil\_get — Retourne les informations sur un groupe d'utilisateurs.

## Synopsis

```
grouputil login_grouputil_get(prm_token, prm_gut_id);
```

```
int4 prm_token;  
int4 prm_gut_id;
```

## Description

Retourne les informations sur un groupe d'utilisateurs.

Entrée :

- *prm\_token* : token d'authentification
- *prm\_gut\_id* : Id du groupe d'utilisateurs

Retour :

- *gut\_id* : Id du groupe
- *gut\_nom* : Nom du groupe d'utilisateurs

Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE  
  ret login.grouputil;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  SELECT * INTO ret FROM login.grouputil WHERE gut_id = prm_gut_id;  
  RETURN ret;  
END;
```

## Name

login\_grouputil\_groupe\_liste — Retourne la liste des groupes d'utilisateurs accessibles par un groupe d'utilisateurs.

## Synopsis

```
setof grouputil_groupe login_grouputil_groupe_liste(prm_token, prm_gut_id);
```

```
int4 prm_token;
```

```
int4 prm_gut_id;
```

## Description

Retourne la liste des groupes d'utilisateurs accessibles par un groupe d'utilisateurs.

Entrée :

- *prm\_token* : token d'authentification
- *prm\_gut\_id* : Id du groupe d'utilisateurs

Retour (multiple) :

- *gut\_id* : Id du groupe d'utilisateurs
- *grp\_id* : Id du groupe d'utilisateurs

Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE
    row login.grouputil_groupe;
BEGIN
    PERFORM login._token_assert (prm_token, TRUE, FALSE);
    FOR row IN SELECT * FROM login.grouputil_groupe WHERE gut_id = prm_gut_id LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`login_grouputil_groupe_set` — Définit la liste des groupes d'utilisateurs accessibles par un groupe d'utilisateurs.

## Synopsis

```
void login_grouputil_groupe_set(prm_token, prm_gut_id, prm_groupes);
```

```
int4 prm_token;
```

```
int4 prm_gut_id;
```

```
int4[] prm_groupes;
```

## Description

Définit la liste des groupes d'utilisateurs accessibles par un groupe d'utilisateurs.

Entrée :

- `prm_token` : token d'authentification
- `prm_gut_id` : Id du groupe d'utilisateurs
- `prm_groupes[]` : tableau d'identifiants de groupes d'utilisateurs

Nécessite les droits à la configuration "Etablissement"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM login.grouputil_groupe WHERE gut_id = prm_gut_id;
  IF prm_groupes NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_groupes, 1) LOOP
      INSERT INTO login.grouputil_groupe(gut_id, grp_id) VALUES (prm_gut_id, prm_groupes[i]);
    END LOOP;
  END IF;
END;
```

## Name

login\_grouputil\_liste — Retourne la liste des groupe d'utilisateurs.

## Synopsis

```
setof grouputil login_grouputil_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des groupe d'utilisateurs.

Entrée :

- `prm_token` : token d'authentification

Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE
    row login.grouputil;
BEGIN
    PERFORM login._token_assert (prm_token, TRUE, FALSE);
    FOR row IN SELECT * FROM login.grouputil ORDER BY gut_nom LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

login\_grouputil\_portail\_liste — Retourne la liste des portails accessibles par un groupe d'utilisateurs.

## Synopsis

```
setof      grouputil_portail      login_grouputil_portail_liste(prm_token,  
prm_gut_id);
```

```
int4 prm_token;  
int4 prm_gut_id;
```

## Description

Retourne la liste des portails accessibles par un groupe d'utilisateurs.

Entrée :

- *prm\_token* : token d'authentification
- *prm\_gut\_id* : Id du groupe d'utilisateurs

Retour (multiple) :

- *gut\_id* : Id du groupe d'utilisateurs
- *por\_id* : Id du portail

Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE  
  row login.grouputil_portail;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  FOR row IN SELECT * FROM login.grouputil_portail WHERE gut_id = prm_gut_id LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```



## Name

`login_grouputil_portail_set` — Définit la liste des portails accessibles par un groupe d'utilisateurs.

## Synopsis

```
void login_grouputil_portail_set(prm_token, prm_gut_id, prm_portails);
```

```
int4 prm_token;
```

```
int4 prm_gut_id;
```

```
int4[] prm_portails;
```

## Description

Définit la liste des portails accessibles par un groupe d'utilisateurs.

Entrée :

- `prm_token` : token d'authentification
- `prm_gut_id` : Id du groupe d'utilisateurs
- `prm_portails[]` : tableau d'identifiants de portails

Nécessite les droits à la configuration "Etablissement"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM login.grouputil_portail WHERE gut_id = prm_gut_id;
  IF prm_portails NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_portails, 1) LOOP
      INSERT INTO login.grouputil_portail(gut_id, por_id) VALUES (prm_gut_id, prm_portails[i]);
    END LOOP;
  END IF;
END;
```

## Name

login\_grouputil\_supprime — Supprime un groupe d'utilisateurs.

## Synopsis

```
void login_grouputil_supprime(prm_token, prm_gut_id);
```

```
int4 prm_token;  
int4 prm_gut_id;
```

## Description

Supprime un groupe d'utilisateurs.

- *prm\_token* : token d'authentification
- *prm\_gut\_id* : Identifiant du groupe d'utilisateurs

Nécessite les droits à la configuration "Etablissement"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  DELETE FROM login.grouputil_groupe WHERE gut_id = prm_gut_id;  
  DELETE FROM login.grouputil_portail WHERE gut_id = prm_gut_id;  
  DELETE FROM login.grouputil WHERE gut_id = prm_gut_id;  
END;
```

## Name

login\_grouputil\_update — Met à jour les informations d'un groupe d'utilisateurs.

## Synopsis

```
void login_grouputil_update(prm_token, prm_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_id;  
varchar prm_nom;
```

## Description

Met à jour les informations d'un groupe d'utilisateurs.

Entrée :

- *prm\_token* : token d'authentification
- *prm\_id* : Identifiant du groupe d'utilisateurs
- *prm\_nom* : nom du groupe à mettre à jour

Nécessite les droits à la configuration "Etablissement"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  UPDATE login.grouputil SET gut_nom = prm_nom WHERE gut_id = prm_id;  
END;
```

## Name

login\_utilisateur\_acces\_personne — Vérifie que le token donne accès à un usager donné.

## Synopsis

```
bool login_utilisateur_acces_personne(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Description

Vérifie que le token donne accès à un usager donné. L'accès est validé si l'usager fait partie d'un groupe d'usagers auquel le groupe d'utilisateurs de l'utilisateur (identifié par le token) a accès.

Entrée :

- `prm_token` : token d'authentification
- `prm_per_id` : Identifiant de l'usager

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  RETURN EXISTS (SELECT 1 FROM login.utilisateur_grouputil
    INNER JOIN login.grouputil_groupe USING (gut_id)
    INNER JOIN personne_groupe USING (grp_id)
    INNER JOIN login.token USING(uti_id)
    WHERE tok_token = prm_token AND per_id = prm_per_id);
END;
```

## Name

utilisateur\_add — Ajoute un utilisateur.

## Synopsis

```
int4  utilisateur_add(prm_token,  prm_login,  prm_per_id,  prm_util_config,
prm_util_root);
```

```
int4  prm_token;
varchar  prm_login;
int4  prm_per_id;
bool  prm_util_config;
bool  prm_util_root;
```

## Description

Ajoute un utilisateur.

Entrée :

- prm\_token : Token d'authentification
- prm\_login : Login du nouvel utilisateur
- prm\_per\_id : Id du personnel lié à l'utilisateur
- prm\_util\_config : l'utilisateur a-t-il droit à la config "Etablissement" ?
- prm\_util\_root : l'utilisateur a-t-il droit à la config "Réseau" ?

Retour :

- L'identifiant de l'utilisateur créé

Remarques : Un mot de passe temporaire est généré. Il peut être connu avec la fonction utilisateur\_get. Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE
  ret integer;
  mdp varchar;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  mdp = LPAD((100000+random()*900000)::varchar, 6, '0');
  INSERT INTO login.utilisateur (uti_login, per_id, uti_salt, uti_pwd, uti_config, uti_root) VALUES (prm_
  RETURN ret;
END;
```

## Name

utilisateur\_get — Retourne les informations d'un utilisateur.

## Synopsis

```
utilisateur utilisateur_get(prm_token, prm_util_id);
```

```
int4 prm_token;  
int4 prm_util_id;
```

## Description

Retourne les informations d'un utilisateur.

- `uti_id` : Id de l'utilisateur
- `uti_login` : login de l'utilisateur, NULL si pas droit de config Établissement
- `uti_salt` : toujours NULL
- `uti_root` : Vrai si l'utilisateur a droit de config "Réseau", NULL si pas droit de config Établissement
- `uti_config` : Vrai si l'utilisateur a droit de config "Établissement", NULL si pas droit de config Établissement
- `per_id` : Id du personnel relié à l'utilisateur
- `uti_pwd` : Mot de passe (si temporaire), NULL si pas droit de config Établissement
- `uti_digest` : toujours NULL

## Source

```
DECLARE  
    ret login.utilisateur;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT utilisateur.* INTO ret FROM login.utilisateur WHERE uti_id = prm_util_id;  
    ret.uti_salt = NULL;  
    ret.uti_digest = NULL;  
    BEGIN  
        PERFORM login._token_assert (prm_token, TRUE, FALSE);  
        RETURN ret;  
        EXCEPTION WHEN insufficient_privilege THEN  
            ret.uti_login = NULL;  
            ret.uti_root = NULL;  
            ret.uti_config = NULL;  
            ret.uti_pwd = NULL;  
            RETURN ret;  
    END;  
END;
```

## Name

utilisateur\_get\_digest\_hash — TODO: utilisé par webdav

## Synopsis

```
vvarchar utilisateur_get_digest_hash(prm_uti_login);
```

```
vvarchar prm_uti_login;
```

## Description

TODO: utilisé par webdav

## Source

```
DECLARE
  ret vvarchar;
BEGIN
  SELECT utilisateur.uti_digest INTO ret FROM login.utilisateur WHERE uti_login = prm_uti_login;
  RETURN ret;
END;
```

## Name

utilisateur\_get\_par\_login — TODO : utilisé par imports temporaires

## Synopsis

```
utilisateur utilisateur_get_par_login(prm_uti_login);
```

```
vvarchar prm_uti_login;
```

## Description

TODO : utilisé par imports temporaires

## Source

```
DECLARE
  ret login.utilisateur;
BEGIN
  SELECT utilisateur.* INTO ret FROM login.utilisateur WHERE uti_login = prm_uti_login;
  RETURN ret;
END;
```



## Name

utilisateur\_groupe\_liste — Retourne la liste des groupes d'utilisateurs accessibles par l'utilisateur authentifié.

## Synopsis

```
setof groupe utilisateur_groupe_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des groupes d'utilisateurs accessibles par l'utilisateur authentifié.

Entrées :

- `prm_token` : Token d'authentification

## Source

```
DECLARE
  row public.groupe;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT groupe.* FROM groupe
      INNER JOIN login.grouputil_groupe USING(grp_id)
      INNER JOIN login.utilisateur_grouputil USING (gut_id)
      INNER JOIN login.token USING(uti_id)
      WHERE tok_token = prm_token
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

utilisateur\_grouputil\_liste — Liste des groupes d'utilisateurs auxquels est affecté un utilisateur.

## Synopsis

```
setof grouputil utilisateur_grouputil_liste(prm_token, prm_util_id);
```

```
int4 prm_token;
```

```
int4 prm_util_id;
```

## Description

Liste des groupes d'utilisateurs auxquels est affecté un utilisateur.

Entrée :

- prm\_token : Token d'authentification
- prm\_util\_id : Id de l'utilisateur concerné

Remarques : Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE
  row login.grouputil;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  FOR row IN
    SELECT grouputil.* FROM login.grouputil
      INNER JOIN login.utilisateur_grouputil USING(gut_id)
      WHERE uti_id = prm_util_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

utilisateur\_grouputil\_set — Modifie les groupes d'utilisateurs auxquels est affecté un utilisateur.

## Synopsis

```
void utilisateur_grouputil_set(prm_token, prm_uti_id, prm_grouputils);
```

```
int4 prm_token;
```

```
int4 prm_uti_id;
```

```
int4[] prm_grouputils;
```

## Description

Modifie les groupes d'utilisateurs auxquels est affecté un utilisateur.

Entrée :

- *prm\_token* : Token d'authentification
- *prm\_uti\_id* : Id de l'Utilisateur à modifier
- *prm\_grouputils* : Tableau d'identifiants de groupes d'utilisateurs

Remarques : Nécessite les droits à la configuration "Etablissement"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM login.utilisateur_grouputil WHERE uti_id = prm_uti_id;
  IF prm_grouputils NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_grouputils, 1) LOOP
      INSERT INTO login.utilisateur_grouputil(uti_id, gut_id) VALUES (prm_uti_id, prm_grouputils[i]);
    END LOOP;
  END IF;
END;
```

## Name

utilisateur\_liste\_details\_configuration — Liste le détail des utilisateurs.

## Synopsis

```
setof                               utilisateur_liste_details_configuration
utilisateur_liste_details_configuration(prm_token);
```

```
int4 prm_token;
```

## Description

Liste le détail des utilisateurs.

Entrée :

- `prm_token` : Token d'authentification

Remarques : Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE
  row login.utilisateur_liste_details_configuration;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  FOR row IN
    SELECT uti_id, uti_login, personne_info_varchar_get (prm_token, per_id, 'prenom'), personne_info_varc
    WHERE per_id NOTNULL
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

utilisateur\_login2 — Authentifie un utilisateur à partir d'un login et mot de passe.

## Synopsis

```
setof utilisateur_login2 utilisateur_login2(prm_login, prm_mdp);
```

```
varchar prm_login;
```

```
varchar prm_mdp;
```

## Description

Authentifie un utilisateur à partir d'un login et mot de passe. Retourne un ensemble de :

- `uti_id` : l'identifiant de l'utilisateur
- `tok_token` : token d'authentification à utiliser dans tout appel de fonction sécurisée
- `uti_root` : l'utilisateur a accès à la configuration du réseau
- `uti_config` : l'utilisateur a accès à la configuration de l'établissement
- `eta_id / por_id` : paire établissement/portail auxquels l'utilisateur a droit
- `uti_pwd_provisoire` (booléen) : indique si le mot de passe est provisoire (stocké en clair en base et visible dans la configuration de l'utilisateur)

Les données autres que `eta_id` et `por_id` sont répétées à l'identique sur toutes les lignes de la réponse.

## Source

```
DECLARE
  row login.utilisateur_login2;
  uti integer;
  tok integer;
BEGIN
  SELECT uti_id INTO uti FROM login.utilisateur
    WHERE uti_login = prm_login AND crypt (prm_mdp, SUBSTRING(uti_salt FROM 1 FOR 2)) = uti_salt;
  IF NOT FOUND THEN
    RETURN;
  END IF;
  SELECT * INTO tok FROM login._token_create (uti);
  IF prm_login = 'kavarna' OR prm_login = 'variation' THEN
    SELECT DISTINCT uti_id, tok, uti_root, uti_config, NULL, NULL, (uti_pwd NOTNULL) INTO row FROM login
      WHERE uti_id = uti;
    RETURN NEXT row;
    RETURN;
  END IF;
  FOR row IN
    select DISTINCT uti_id, tok, uti_root, uti_config, groupe.eta_id, por_id, (uti_pwd NOTNULL) FROM login
      INNER JOIN login.utilisateur_grouputil USING(uti_id)
      INNER JOIN login.grouputil USING (gut_id)
      INNER JOIN login.grouputil_portail USING(gut_id)
      INNER JOIN login.grouputil_groupe USING (gut_id)
      INNER JOIN meta.portail USING(por_id)
      INNER JOIN groupe USING(grp_id)
      INNER JOIN etablissement USING(eta_id)
    WHERE
      uti_id = uti
      AND portail.cat_id = etablissement.cat_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

utilisateur\_login\_digest

## Synopsis

```
int4 utilisateur_login_digest(prm_login, prm_hash, prm_response);
```

```
varchar prm_login;  
varchar prm_hash;  
varchar prm_response;
```

## Source

```
DECLARE  
  digest varchar;  
  myresponse varchar;  
  tok integer;  
BEGIN  
  SELECT uti_digest INTO digest FROM login.utilisateur WHERE uti_login = prm_login;  
  IF NOT FOUND THEN  
    RETURN NULL;  
  END IF;  
  SELECT md5 (digest || ':' || prm_hash) INTO myresponse;  
  IF myresponse = prm_response THEN  
    SELECT * INTO tok FROM login._token_create ((SELECT uti_id FROM login.utilisateur WHERE uti_login  
  RETURN tok;  
  ELSE  
    RETURN NULL;  
  END IF;  
END;
```

## Name

utilisateur\_mdp\_change — Modifie le mot de passe de l'utilisateur identifié.

## Synopsis

```
void utilisateur_mdp_change(prm_token, prm_mdp);
```

```
int4 prm_token;  
varchar prm_mdp;
```

## Description

Modifie le mot de passe de l'utilisateur identifié.

Entrée :

- *prm\_token* : Token d'authentification
- *prm\_mdp* : Nouveau mot de passe

Remarques : Seul l'utilisateur connecté peut modifier son mot de passe. Il est possible pour un utilisateur ayant les droits de configuration "Etablissement" de générer un mot de passe temporaire avec `utilisateur_mdp_genere` pour tout autre utilisateur, mais seul l'utilisateur lui-même peut saisir un nouveau mot de passe.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE login.utilisateur SET uti_salt = crypt (prm_mdp, gen_salt('des')), uti_pwd = NULL  
    WHERE uti_id = (SELECT uti_id FROM login.token WHERE tok_token = prm_token);  
  UPDATE login.utilisateur SET uti_digest = md5 (uti_login || ':Accueil:' || prm_mdp)  
    WHERE uti_id = (SELECT uti_id FROM login.token WHERE tok_token = prm_token);  
END;
```

## Name

utilisateur\_mdp\_genere — Génère un nouveau mot de passe aléatoire pour un utilisateur.

## Synopsis

```
void utilisateur_mdp_genere(prm_token, prm_uti_id);
```

```
int4 prm_token;  
int4 prm_uti_id;
```

## Description

Génère un nouveau mot de passe aléatoire pour un utilisateur.

Entrée :

- *prm\_token* : Token d'authentification
- *prm\_uti\_id* : Id de l'utilisateur pour qui générer un nouveau mot de passe

Remarques : Nécessite les droits à la configuration "Etablissement"

## Source

```
DECLARE  
    mdp varchar;  
BEGIN  
    PERFORM login._token_assert (prm_token, TRUE, FALSE);  
    mdp = LPAD((random()*1000000)::int::varchar, 6, '0');  
    UPDATE login.utilisateur SET uti_salt = crypt (mdp, gen_salt('des')), uti_pwd = mdp  
        WHERE uti_id = prm_uti_id;  
    UPDATE login.utilisateur SET uti_digest = md5 (uti_login || ':Accueil:' || uti_pwd)  
        WHERE uti_id = prm_uti_id;  
END;
```



## Name

utilisateur\_portail\_liste — TODO : Utilisé par webdav

## Synopsis

```
setof int4 utilisateur_portail_liste(prm_login);
```

```
varchar prm_login;
```

## Description

TODO : Utilisé par webdav

## Source

```
DECLARE
  row RECORD;
BEGIN
  FOR row IN
    select DISTINCT por_id FROM login.utilisateur
    INNER JOIN login.utilisateur_grouputil USING(uti_id)
    INNER JOIN login.grouputil USING (gut_id)
    INNER JOIN login.grouputil_portail USING(gut_id)
    INNER JOIN login.grouputil_groupe USING (gut_id)
    INNER JOIN groupe USING(grp_id)
    WHERE uti_login = prm_login
  LOOP
    RETURN NEXT row.por_id;
  END LOOP;
END;
```

## Name

utilisateur\_prenom\_nom — Retourne le nom et prénom d'un utilisateur

## Synopsis

```
vvarchar utilisateur_prenom_nom(prm_token, prm_uti_id);
```

```
int4 prm_token;
```

```
int4 prm_uti_id;
```

## Description

Retourne le nom et prénom d'un utilisateur

## Source

```
DECLARE
  ret varchar;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT personne_get_libelle (prm_token, per_id) INTO ret FROM personne INNER JOIN login.utilisateur USI
  RETURN ret;
END;
```

## Name

utilisateur\_supprime — Supprime un utilisateur.

## Synopsis

```
void utilisateur_supprime(prm_token, prm_uti_id);
```

```
int4 prm_token;  
int4 prm_uti_id;
```

## Description

Supprime un utilisateur.

Entrée :

- *prm\_token* : Token d'authentification
- *prm\_uti\_id* : Id de l'utilisateur à supprimer

Remarques : Nécessite les droits à la configuration "Etablissement"

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  DELETE FROM login.utilisateur_grouputil WHERE uti_id = prm_uti_id;  
  DELETE FROM login.utilisateur WHERE uti_id = prm_uti_id;  
END;
```

## Name

utilisateur\_update — Met à jour les informations d'un utilisateur.

## Synopsis

```
void utilisateur_update(prm_token, prm_uti_id, prm_login, prm_per_id,  
prm_uti_config, prm_uti_root);
```

```
int4 prm_token;  
int4 prm_uti_id;  
varchar prm_login;  
int4 prm_per_id;  
bool prm_uti_config;  
bool prm_uti_root;
```

## Description

Met à jour les informations d'un utilisateur.

Entrée :

- prm\_token : Token d'authentification
- prm\_uti\_id : Id de l'utilisateur à mettre à jour
- prm\_login : Nouveau login
- prm\_per\_id : Nouveau lien vers personnel
- prm\_uti\_config : Nouveau droit de configuration "Établissement"
- prm\_uti\_root : Nouveau droit de configuration "Réseau"

## Source

```
BEGIN  
  PERFORM login.token_assert (prm_token, TRUE, FALSE);  
  UPDATE login.utilisateur SET uti_login = prm_login, per_id = prm_per_id, uti_config = prm_uti_config, u  
END;
```

## Name

utilisateur\_usagers\_liste — Retourne la liste des usagers en relation avec l'utilisateur authentifié par le token.

## Synopsis

```
setof      utilisateur_usagers_liste      utilisateur_usagers_liste(prm_token,
prm_grp_id, prm_presents);
```

```
int4 prm_token;
int4 prm_grp_id;
bool prm_presents;
```

## Description

Retourne la liste des usagers en relation avec l'utilisateur authentifié par le token.

Entrée :

- *prm\_token* : token d'authentification
- *prm\_grp\_id* : Identifiant du groupe auquel doivent appartenir les usagers, ou NULL pour rechercher parmi tous les groupes
- *prm\_presents* : TRUS pour retourner les usagers présents uniquement, FALSE sinon

## Source

```
DECLARE
    uti integer;
    row login.utilisateur_usagers_liste;
BEGIN
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    FOR row IN
        SELECT DISTINCT personne.per_id, personne_get_libelle (prm_token, personne.per_id)
        FROM personne
        inner join personne_groupe using(per_id)
        inner join groupe_secteur using(grp_id)
        inner join meta.secteur using(sec_id)
        inner join login.grouputil_groupe USING(grp_id)
        inner join login.utilisateur_grouputil using(gut_id)
        where uti_id = uti
        -- and sec_code = 'prise_en_charge'
        and ent_code = 'usager'
        AND (prm_grp_id ISNULL OR grp_id = prm_grp_id)
        and (prm_presents ISNULL OR NOT prm_presents OR personne_info_integer_get (prm_token, per_id, 'stat
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## 10. meta

### 10.1. Description

Informations de construction de l'interface. Informations de base.

### 10.2. Tables

#### 10.2.1. meta.categorie

Catégorie d'établissement

cat\_id (int4)

cat\_nom (varchar)

cat\_code (varchar)

### **Liens vers cette table**

- portail.cat\_id
- etablissement.cat\_id

### **10.2.2. meta.dirinfo**

Banque de champs : répertoire contenant les champs

din\_id (int4)

din\_id\_parent (int4)

Clé étrangère vers dirinfo.din\_id

din\_libelle (varchar)

### **Liens vers cette table**

- dirinfo.din\_id\_parent
- info.din\_id

### **10.2.3. meta.entite**

Type de personne

ent\_id (int4)

ent\_code (varchar)

ent\_libelle (varchar)

### **Liens vers cette table**

- liste.ent\_id
- menu.ent\_id
- metier\_entite.ent\_id
- droit\_ajout\_entite\_portail.ent\_code
- personne.ent\_code

### **10.2.4. meta.groupe\_infos**

Groupe de champs dans un sous-menu (édition de personnes)

gin\_id (int4)

sme\_id (int4)

Clé étrangère vers sousmenu.sme\_id

gin\_libelle (varchar)

gin\_ordre (int4)

### **Liens vers cette table**

- info\_groupe.gin\_id

#### **10.2.5. meta.info**

Champ d'édition d'une personne

inf\_id (int4)

int\_id (int4)

Clé étrangère vers infos\_type.int\_id

inf\_code (varchar)

inf\_libelle (varchar)

inf\_\_textelong\_nblignes (int4)

inf\_\_selection\_code (int4)

Clé étrangère vers selection.sel\_id

inf\_etendu (bool)

inf\_historique (bool)

inf\_multiple (bool)

inf\_\_groupe\_type (varchar)

inf\_\_contact\_filtre (varchar)

inf\_\_metier\_secteur (varchar)

inf\_\_contact\_secteur (varchar)

inf\_\_etablissement\_interne (bool)

din\_id (int4)

Clé étrangère vers dirinfo.din\_id

inf\_\_groupe\_soustype (int4)

inf\_libelle\_complet (varchar)

inf\_\_date\_echeance (bool)

inf\_\_date\_echeance\_icone (varchar)

inf\_\_date\_echeance\_secteur (varchar)

Clé étrangère vers secteur.sec\_code

inf\_\_etablissement\_secteur (varchar)

Clé étrangère vers secteur.sec\_code

## Liens vers cette table

- champ.inf\_id
- supp.inf\_id
- info\_aide.inf\_id
- info\_groupe.inf\_id
- groupe\_info\_secteur.inf\_id
- personne\_groupe.\_inf\_id
- personne\_info.inf\_code

### 10.2.6. meta.info\_aide

Texte d'aide d'un champ

ina\_id (int4)

inf\_id (int4)

Clé étrangère vers info.inf\_id

ina\_aide (text)

### 10.2.7. meta.info\_groupe

Assignation d'un champ dans un groupe de champs

ing\_id (int4)

inf\_id (int4)

Clé étrangère vers info.inf\_id

gin\_id (int4)

Clé étrangère vers groupe\_infos.gin\_id

ing\_ordre (int4)

ing\_\_groupe\_cycle (bool)

ing\_obligatoire (bool)

### 10.2.8. meta.infos\_type

Types de champs (édition des personnes)

int\_id (int4)

int\_code (varchar)

int\_libelle (varchar)

int\_multiple (bool)

int\_historique (bool)



### **Liens vers cette table**

- info.int\_id

#### **10.2.9. meta.lien\_familial**

Types de liens familiaux

lfa\_id (int4)

lfa\_nom (varchar)

### **Liens vers cette table**

- personne\_info\_lien\_familial.lfa\_id

#### **10.2.10. meta.menu**

Menu du dialogue d'édition d'une personne

men\_id (int4)

men\_libelle (varchar)

men\_ordre (int4)

ent\_id (int4)

Clé étrangère vers entite.ent\_id

por\_id (int4)

Clé étrangère vers portail.por\_id

### **Liens vers cette table**

- sousmenu.men\_id

#### **10.2.11. meta.metier**

Liste des métiers

met\_id (int4)

met\_nom (varchar)

### **Liens vers cette table**

- metier\_entite.met\_id
- metier\_secteur.met\_id

#### **10.2.12. meta.metier\_entite**

Affectation d'un métier à un type de personne

mee\_id (int4)

met\_id (int4)

Clé étrangère vers metier.met\_id

ent\_id (int4)

Clé étrangère vers entite.ent\_id

### 10.2.13. meta.metier\_secteur

Asignation d'un métier à un secteur

mes\_id (int4)

met\_id (int4)

Clé étrangère vers metier.met\_id

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

### 10.2.14. meta.portail

Liste des portails

por\_id (int4)

cat\_id (int4)

Clé étrangère vers categorie.cat\_id

por\_libelle (varchar)

#### Liens vers cette table

- grouputil\_portail.por\_id
- menu.por\_id
- topmenu.por\_id
- theme\_portail.por\_id
- droit\_ajout\_entite\_portail.por\_id

### 10.2.15. meta.secteur

Liste des secteurs de métiers

sec\_id (int4)

sec\_code (varchar)

sec\_nom (varchar)

sec\_est\_prise\_en\_charge (bool)

sec\_icone (varchar)

#### Liens vers cette table

- document\_secteur.sec\_id
- document\_type\_secteur.sec\_id

- documents\_secteur.sec\_id
- event\_type\_secteur.sec\_id
- secteur\_event.sec\_id
- secteur\_events.sec\_id
- localisation\_secteur.sec\_id
- info.inf\_\_date\_echeance\_secteur
- info.inf\_\_etablissement\_secteur
- metier\_secteur.sec\_id
- secteur\_type.sec\_id
- etablissement\_secteur.sec\_id
- etablissement\_secteur\_edit.sec\_id
- groupe\_info\_secteur.sec\_id
- groupe\_secteur.sec\_id
- secteur\_infos.sec\_id
- ressource\_secteur.sec\_id

### **10.2.16. meta.secteur\_type**

Sous-catégorie des secteurs

set\_id (int4)

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

set\_nom (varchar)

### **10.2.17. meta.selection**

Liste des sélection (pour les champs sélection d'édition des personnes)

sel\_id (int4)

sel\_code (varchar)

sel\_libelle (varchar)

sel\_info (varchar)

### **Liens vers cette table**

- info.inf\_\_selection\_code
- selection\_entree.sel\_id

### **10.2.18. meta.selection\_entree**

Valeurs des listes de sélection

sen\_id (int4)

sel\_id (int4)

Clé étrangère vers selection.sel\_id

sen\_libelle (varchar)

sen\_ordre (int4)

### **10.2.19. meta.sousmenu**

Sous-menu du dialogue d'édition d'une personne

sme\_id (int4)

men\_id (int4)

Clé étrangère vers menu.men\_id

sme\_libelle (varchar)

sme\_ordre (int4)

sme\_type (varchar)

sme\_type\_id (int4)

sme\_droit\_modif (bool)

sme\_droit\_suppression (bool)

sme\_icone (varchar)

#### **Liens vers cette table**

- groupe\_infos.sme\_id
- procedure\_affectation.sme\_id

### **10.2.20. meta.topmenu**

Menu principal

tom\_id (int4)

tom\_libelle (varchar)

tom\_ordre (int4)

por\_id (int4)

Clé étrangère vers portail.por\_id

#### **Liens vers cette table**

- topsousmenu.tom\_id

### **10.2.21. meta.topsousmenu**

Sous-menus du menu principal

tsm\_id (int4)

tom\_id (int4)

Clé étrangère vers topmenu.tom\_id

tsm\_libelle (varchar)

tsm\_ordre (int4)

tsm\_icone (varchar)

tsm\_script (varchar)

tsm\_type (varchar)

tsm\_type\_id (int4)

tsm\_titre (varchar)

tsm\_droit\_modif (bool)

tsm\_droit\_suppression (bool)

sme\_id\_lien\_usager (int4)

### **Liens vers cette table**

- `procedure_affectation.tsm_id`

## **10.3. Types**

### **10.3.1. meta.meta\_info\_groupe\_liste**

### **10.3.2. meta.meta\_sousmenus\_liste\_depuis\_topmenu**

### **10.3.3. meta.metier\_liste\_details**

### **10.3.4. meta.meta\_info\_usage**

## **10.4. Fonctions**

## Name

`meta_categorie_add` — Ajoute une nouvelle catégorie d'établissement.

## Synopsis

```
int4 meta_categorie_add(prm_token, prm_nom, prm_code);
```

```
int4 prm_token;  
varchar prm_nom;  
varchar prm_code;
```

## Description

Ajoute une nouvelle catégorie d'établissement. Entrées :

- `prm_token` : Token d'authentification
- `prm_nom` : Nom de la catégorie
- `prm_code` : Code de la catégorie, ou NULL pour une affectation automatique selon le nom

## Source

```
DECLARE  
    ret integer;  
    code varchar;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO meta.categorie (cat_nom, cat_code)  
        VALUES (prm_nom, COALESCE (prm_code, pour_code(prm_nom)))  
    RETURNING cat_id INTO ret;  
    RETURN ret;  
END;
```

## Name

`meta_categorie_delete` — Supprime une catégorie de manière non récursive (les portails de la catégorie doivent être supprimés auparavant).

## Synopsis

```
void meta_categorie_delete(prm_token, prm_cat_id);
```

```
int4 prm_token;
```

```
int4 prm_cat_id;
```

## Description

Supprime une catégorie de manière non récursive (les portails de la catégorie doivent être supprimés auparavant).

## Source

```
BEGIN
-- DELETE FROM meta.info_groupe WHERE gin_id IN (SELECT gin_id FROM meta.groupe_infos WHERE sme_id IN (S
-- DELETE FROM meta.groupe_infos WHERE sme_id IN (SELECT sme_id FROM meta.sousmenu WHERE men_id IN (SELE
-- DELETE FROM meta.sousmenu WHERE men_id IN (SELECT men_id FROM meta.menu WHERE por_id IN (SELECT por_i
-- DELETE FROM meta.menu WHERE por_id IN (SELECT por_id FROM meta.portail WHERE cat_id = prm_cat_id);

-- DELETE FROM meta.topsousmenu WHERE tom_id IN (SELECT tom_id FROM meta.topmenu WHERE por_id IN (SELECT
-- DELETE FROM meta.topmenu WHERE por_id IN (SELECT por_id FROM meta.portail WHERE cat_id = prm_cat_id);

-- DELETE FROM meta.portail WHERE cat_id = prm_cat_id;

PERFORM login._token_assert (prm_token, FALSE, FALSE);
PERFORM login._token_assert_interface (prm_token);
DELETE FROM meta.categorie WHERE cat_id = prm_cat_id;
END;
```

## Name

meta\_categorie\_liste — Retourne la liste des catégories.

## Synopsis

```
setof categorie meta_categorie_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des catégories.

## Source

```
DECLARE
    row meta.categorie;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.categorie ORDER BY cat_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```



## Name

meta\_categorie\_rename — Renomme une catégorie.

## Synopsis

```
void meta_categorie_rename(prm_token, prm_cat_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_cat_id;  
varchar prm_nom;
```

## Description

Renomme une catégorie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.categorie SET cat_nom = prm_nom WHERE cat_id = prm_cat_id;  
END;
```

## Name

`meta_dirinfo_add_avec_id` — Ajoute un nouveau répertoire de champs (utilisé avec la banque de champs).

## Synopsis

```
int4    meta_dirinfo_add_avec_id(prm_token,    prm_id,    prm_din_id_parent,  
prm_libelle);
```

```
int4 prm_token;  
int4 prm_id;  
int4 prm_din_id_parent;  
varchar prm_libelle;
```

## Description

Ajoute un nouveau répertoire de champs (utilisé avec la banque de champs).

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO meta.dirinfo (din_id, din_id_parent, din_libelle) VALUES (prm_id, prm_din_id_parent, prm_li  
        RETURNING din_id INTO ret;  
    PERFORM setval ('meta.dirinfo_din_id_seq', coalesce((select max(din_id)+1 from meta.dirinfo), 1), false  
    RETURN ret;  
END;
```

## Name

`meta_dirinfo_dernier` — Retourne l'identifiant du dernier répertoire de champ présent (utilisé avec la banque de champs).

## Synopsis

```
int4 meta_dirinfo_dernier(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne l'identifiant du dernier répertoire de champ présent (utilisé avec la banque de champs).

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT MAX(din_id) INTO ret FROM meta.dirinfo;
  RETURN ret;
END;
```

## Name

`meta_dirinfo_list` — Retourne la liste des répertoires de champs inclus dans un répertoire donné.

## Synopsis

```
setof dirinfo meta_dirinfo_list(prm_token, prm_din_id_parent);
```

```
int4 prm_token;
```

```
int4 prm_din_id_parent;
```

## Description

Retourne la liste des répertoires de champs inclus dans un répertoire donné.

## Source

```
DECLARE
    row meta.dirinfo;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row IN
        SELECT * FROM meta.dirinfo WHERE (prm_din_id_parent ISNULL AND din_id_parent ISNULL) OR (din_id_paren
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

meta\_entite\_infos\_par\_code — Retourne les informations sur un type de personne.

## Synopsis

```
entite meta_entite_infos_par_code(prm_token, prm_code);
```

```
int4 prm_token;  
varchar prm_code;
```

## Description

Retourne les informations sur un type de personne.

## Source

```
DECLARE  
    ret meta.entite%ROWTYPE;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM meta.entite WHERE ent_code = prm_code;  
    RETURN ret;  
END;
```

## Name

meta\_entite\_liste — Retourne la liste des types de personnes.

## Synopsis

```
setof entite meta_entite_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des types de personnes.

## Source

```
DECLARE
    row RECORD;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.entite ORDER BY ent_id
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

meta\_groupe\_infos\_add\_end — Ajoute un groupe de champs à la fin d'une page de champs.

## Synopsis

```
int4 meta_groupe_infos_add_end(prm_token, prm_sme_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

```
varchar prm_libelle;
```

## Description

Ajoute un groupe de champs à la fin d'une page de champs.

## Source

```
DECLARE
  int_ordre integer;
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT MAX(gin_ordre) + 1 INTO int_ordre FROM meta.groupe_infos WHERE sme_id = prm_sme_id;
  IF int_ordre ISNULL THEN int_ordre = 0; END IF;
  INSERT INTO meta.groupe_infos(sme_id, gin_libelle, gin_ordre) VALUES (prm_sme_id, prm_libelle, int_ordre);
  RETURN ret;
END;
```

## Name

meta\_groupe\_infos\_delete — Supprime un groupe de champs.

## Synopsis

```
void meta_groupe_infos_delete(prm_token, prm_gin_id);
```

```
int4 prm_token;
```

```
int4 prm_gin_id;
```

## Description

Supprime un groupe de champs.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login.token_assert_interface (prm_token);
  DELETE FROM meta.info_groupe WHERE gin_id = prm_gin_id;
  DELETE FROM meta.groupe_infos WHERE gin_id = prm_gin_id;
END;
```



## Name

meta\_groupe\_infos\_liste — Retourne la liste des groupes de champs d'une page de champs.

## Synopsis

```
setof groupe_infos meta_groupe_infos_liste(prm_token, prm_sme_id);
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

## Description

Retourne la liste des groupes de champs d'une page de champs.

## Source

```
DECLARE
  row meta.groupe_infos%ROWTYPE;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT * FROM meta.groupe_infos WHERE sme_id = prm_sme_id ORDER BY gin_ordre
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

meta\_groupe\_infos\_update — Modifie l'ordre d'apparition d'un groupe de champs.

## Synopsis

```
void meta_groupe_infos_update(prm_token, prm_gin_id, prm_ordre);
```

```
int4 prm_token;  
int4 prm_gin_id;  
int4 prm_ordre;
```

## Description

Modifie l'ordre d'apparition d'un groupe de champs.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.groupe_infos SET  
    gin_ordre = prm_ordre  
  WHERE gin_id = prm_gin_id;  
END;
```

## Name

`meta_info_add_avec_id` — Ajoute un champ dans la bibliothèque de champs disponibles.

## Synopsis

```
int4      meta_info_add_avec_id(prm_token,      prm_inf_id,      prm_int_id,
prm_code,  prm_libelle,  prm__textelong_nblignes,  prm__selection_code,
prm_etendu,  prm_historique,  prm_multiple,  prm__groupe_type,
prm__contact_filtre,  prm__metier_secteur,  prm__contact_secteur,
prm__etablissement_interne,  prm_din_id,  prm__groupe_soustype,
prm_libelle_complet,  prm__date_echeance,  prm__date_echeance_icone,
prm__date_echeance_secteur,  prm__etablissement_secteur);
```

```
int4 prm_token;
int4 prm_inf_id;
int4 prm_int_id;
varchar prm_code;
varchar prm_libelle;
int4 prm__textelong_nblignes;
int4 prm__selection_code;
bool prm_etendu;
bool prm_historique;
bool prm_multiple;
varchar prm__groupe_type;
varchar prm__contact_filtre;
varchar prm__metier_secteur;
varchar prm__contact_secteur;
bool prm__etablissement_interne;
int4 prm_din_id;
int4 prm__groupe_soustype;
varchar prm_libelle_complet;
bool prm__date_echeance;
varchar prm__date_echeance_icone;
varchar prm__date_echeance_secteur;
varchar prm__etablissement_secteur;
```

## Description

Ajoute un champ dans la bibliothèque de champs disponibles.

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  INSERT INTO meta.info(inf_id, int_id, inf_code, inf_libelle, inf__textelong_nblignes, inf__selection_co
  PERFORM setval ('meta.info_inf_id_seq', coalesce((select max(inf_id)+1 from meta.info), 1), false);
  RETURN ret;
END;
```

## Name

`meta_info_aide_get` — Retourne le message d'aide d'un champ.

## Synopsis

```
text meta_info_aide_get(prm_token, prm_inf_id);
```

```
int4 prm_token;
```

```
int4 prm_inf_id;
```

## Description

Retourne le message d'aide d'un champ.

## Source

```
DECLARE
  ret text;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT ina_aide INTO ret FROM meta.info_aide WHERE inf_id = prm_inf_id;
  RETURN ret;
END;
```

## Name

meta\_info\_aide\_set — Modifie le message d'aide d'un champ.

## Synopsis

```
void meta_info_aide_set(prm_token, prm_inf_id, prm_aide);
```

```
int4 prm_token;  
int4 prm_inf_id;  
text prm_aide;
```

## Description

Modifie le message d'aide d'un champ.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  IF prm_aide ISNULL THEN  
    DELETE FROM meta.info_aide WHERE inf_id = prm_inf_id;  
  ELSE  
    UPDATE meta.info_aide SET ina_aide = prm_aide WHERE inf_id = prm_inf_id;  
    IF NOT FOUND THEN  
      INSERT INTO meta.info_aide (inf_id, ina_aide) VALUES (prm_inf_id, prm_aide);  
    END IF;  
  END IF;  
END;
```

## Name

`meta_info_dernier` — Retourne l'identifiant du dernier champ dans la bibliothèque de champs locale.

## Synopsis

```
int4 meta_info_dernier(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne l'identifiant du dernier champ dans la bibliothèque de champs locale.

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT MAX(inf_id) INTO ret FROM meta.info;
  RETURN ret;
END;
```

## Name

`meta_info_get` — Retourne les informations sur un champ.

## Synopsis

```
info meta_info_get(prm_token, prm_inf_id);
```

```
int4 prm_token;
```

```
int4 prm_inf_id;
```

## Description

Retourne les informations sur un champ.

## Source

```
DECLARE
  ret meta.info;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.info WHERE inf_id = prm_inf_id;
  RETURN ret;
END;
```

## Name

`meta_info_get_par_code` — Retourne les informations sur un champ, à partir de son code.

## Synopsis

```
info meta_info_get_par_code(prm_token, prm_inf_code);
```

```
int4 prm_token;
```

```
varchar prm_inf_code;
```

## Description

Retourne les informations sur un champ, à partir de son code.

## Source

```
DECLARE
  ret meta.info;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.info WHERE inf_code = prm_inf_code;
  RETURN ret;
END;
```



## Name

`meta_info_get_type_par_code` — Retourne le type d'un champ.

## Synopsis

```
vvarchar meta_info_get_type_par_code(prm_token, prm_code);
```

```
int4 prm_token;  
vvarchar prm_code;
```

## Description

Retourne le type d'un champ.

## Source

```
DECLARE  
    ret vvarchar;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT int_code INTO ret FROM meta.info  
        INNER JOIN meta.infos_type USING(int_id)  
        WHERE inf_code = prm_code;  
    RETURN ret;  
END;
```

## Name

`meta_info_groupe_add_by_id` — Ajoute un champ à une page (en le rajoutant à un groupe de champs).

## Synopsis

```
int4    meta_info_groupe_add_by_id(prm_token,    prm_inf_id,    prm_gin_id,  
prm_gin_ordre);
```

```
int4 prm_token;  
int4 prm_inf_id;  
int4 prm_gin_id;  
int4 prm_gin_ordre;
```

## Description

Ajoute un champ à une page (en le rajoutant à un groupe de champs).

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO meta.info_groupe (inf_id, gin_id, ing_ordre) VALUES  
        (prm_inf_id, prm_gin_id, prm_gin_ordre)  
    RETURNING ing_id INTO ret;  
    RETURN ret;  
END;
```

## Name

`meta_info_groupe_add_end` — Ajoute un champ à un groupe de champs, en le plaçant à la fin.

## Synopsis

```
int4    meta_info_groupe_add_end(prm_token,    prm_inf_code,    prm_gin_id,  
prm_groupe_cycle);
```

```
int4 prm_token;  
varchar prm_inf_code;  
int4 prm_gin_id;  
bool prm_groupe_cycle;
```

## Description

Ajoute un champ à un groupe de champs, en le plaçant à la fin.

## Source

```
DECLARE  
    ret integer;  
    int_ordre integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    SELECT MAX(ing_ordre) + 1 INTO int_ordre FROM meta.info_groupe WHERE gin_id = prm_gin_id;  
    IF int_ordre ISNULL THEN int_ordre = 0; END IF;  
    INSERT INTO meta.info_groupe (inf_id, gin_id, ing_ordre, ing_groupe_cycle) VALUES  
        ((SELECT inf_id FROM meta.info WHERE inf_code = prm_inf_code), prm_gin_id, int_ordre, prm_groupe_cyc)  
    RETURNING ing_id INTO ret;  
    RETURN ret;  
END;
```

## Name

`meta_info_groupe_delete` — Retire un champ d'un groupe de champs.

## Synopsis

```
void meta_info_groupe_delete(prm_token, prm_ing_id);
```

```
int4 prm_token;
```

```
int4 prm_ing_id;
```

## Description

Retire un champ d'un groupe de champs.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM meta.info_groupe where ing_id = prm_ing_id;
END;
```

## Name

`meta_info_groupe_get` — Retourne les caractéristiques de l'affectation d'un champ à un groupe.

## Synopsis

```
info_groupe meta_info_groupe_get(prm_token, prm_ing_id);
```

```
int4 prm_token;
```

```
int4 prm_ing_id;
```

## Description

Retourne les caractéristiques de l'affectation d'un champ à un groupe.

## Source

```
DECLARE
    row meta.info_groupe;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO row FROM meta.info_groupe WHERE ing_id = prm_ing_id;
    RETURN row;
END;
```

## Name

`meta_info_groupe_liste` — Retourne les informations sur les champs affectés à un groupe de champs (dont les caractéristiques d'affectation).

## Synopsis

```
setof meta_info_groupe_liste meta_info_groupe_liste(prm_token, prm_gin_id);
```

```
int4 prm_token;  
int4 prm_gin_id;
```

## Description

Retourne les informations sur les champs affectés à un groupe de champs (dont les caractéristiques d'affectation).

## Source

```
DECLARE  
  row meta.meta_info_groupe_liste%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT info_groupe.ing_id, info_groupe.ing_ordre, info_groupe.ing__groupe_cycle, info.* FROM meta.inf  
    INNER JOIN meta.info_groupe USING(inf_id) WHERE gin_id = prm_gin_id ORDER BY ing_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`meta_info_groupe_obligatoire_liste` — Retourne les informations sur les champs obligatoires à la création d'une personne d'un type donné sur un portail donné.

## Synopsis

```
setof meta_info_groupe_liste meta_info_groupe_obligatoire_liste(prm_token,  
prm_por_id, prm_ent_code);
```

```
int4 prm_token;  
int4 prm_por_id;  
varchar prm_ent_code;
```

## Description

Retourne les informations sur les champs obligatoires à la création d'une personne d'un type donné sur un portail donné.

## Source

```
DECLARE  
  row meta.meta_info_groupe_liste%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT info_groupe.ing_id, info_groupe.ing_ordre, info_groupe.ing_groupe_cycle, info.*  
      FROM meta.info  
      INNER JOIN meta.info_groupe USING(inf_id)  
      INNER JOIN meta.groupe_infos USING(gin_id)  
      INNER JOIN meta.sousmenu USING(sme_id)  
      INNER JOIN meta.menu USING(men_id)  
      INNER JOIN meta.entite USING(ent_id)  
      WHERE por_id = prm_por_id AND ent_code = prm_ent_code AND ing_obligatoire ORDER BY men_ordre, sme_o  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`meta_info_groupe_update` — Modifie les informations d'affectation d'un champ à un groupe.

## Synopsis

```
void meta_info_groupe_update(prm_token, prm_ing_id, prm_gin_id, prm_ordre);
```

```
int4 prm_token;  
int4 prm_ing_id;  
int4 prm_gin_id;  
int4 prm_ordre;
```

## Description

Modifie les informations d'affectation d'un champ à un groupe.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info_groupe SET gin_id = prm_gin_id, ing_ordre = prm_ordre WHERE ing_id = prm_ing_id;  
END;
```



## Name

`meta_info_liste` — Retourne la liste des champs dans le nom ou le code contient une chaîne, parmi tous les champs disponibles ou seulement ceux utilisés.

## Synopsis

```
setof info meta_info_liste(prm_token, str, usedonly, prm_int_code);
```

```
int4 prm_token;
varchar str;
bool usedonly;
varchar prm_int_code;
```

## Description

Retourne la liste des champs dans le nom ou le code contient une chaîne, parmi tous les champs disponibles ou seulement ceux utilisés.

## Source

```
DECLARE
  row meta.info;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  IF usedonly THEN
    FOR row IN
      SELECT DISTINCT info.* FROM meta.info
        INNER JOIN meta.info_groupe USING(inf_id)
        INNER JOIN meta.infos_type USING(int_id)
        WHERE (str ISNULL OR inf_code ilike '%'||str||%' OR inf_libelle ilike '%'||str||%')
        AND (prm_int_code ISNULL OR prm_int_code = infos_type.int_code)
        ORDER BY inf_libelle
    LOOP
      RETURN NEXT row;
    END LOOP;
  ELSE
    FOR row IN
      SELECT info.* FROM meta.info
        INNER JOIN meta.infos_type USING(int_id)
        WHERE (str ISNULL OR inf_code ilike '%'||str||%' OR inf_libelle ilike '%'||str||%')
        AND (prm_int_code ISNULL OR prm_int_code = infos_type.int_code)
        ORDER BY inf_libelle
    LOOP
      RETURN NEXT row;
    END LOOP;
  END IF;
END;
```

## Name

`meta_info_liste_champs_par_secteur_categorie` — Retourne la liste des champs de type "groupe" couvrant le secteur donné affectés dans une fiche du portail

## Synopsis

```
setof info meta_info_liste_champs_par_secteur_categorie(prm_token,  
prm_cat_id, prm_sec_code);
```

```
int4 prm_token;  
int4 prm_cat_id;  
varchar prm_sec_code;
```

## Description

Retourne la liste des champs de type "groupe" couvrant le secteur donné affectés dans une fiche du portail

## Source

```
DECLARE  
  row meta.info;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    select distinct info.* FROM meta.info  
    inner join meta.infos_type using(int_id)  
    inner join meta.info_groupe using(inf_id)  
    inner join meta.groupe_infos using(gin_id)  
    inner join meta.sousmenu using(sme_id)  
    inner join meta.menu using(men_id)  
    inner join meta.portail using(por_id)  
    where int_code = 'groupe'  
    AND cat_id = prm_cat_id  
    AND inf__groupe_type = prm_sec_code  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

meta\_info\_update — Modifie les informations d'un champ.

## Synopsis

```
void meta_info_update(prm_token, prm_inf_id, prm_int_id, prm_code,  
prm_libelle, prm_libelle_complet, prm_etendu, prm_historique, prm_multiple);
```

```
int4 prm_token;  
int4 prm_inf_id;  
int4 prm_int_id;  
varchar prm_code;  
varchar prm_libelle;  
varchar prm_libelle_complet;  
bool prm_etendu;  
bool prm_historique;  
bool prm_multiple;
```

## Description

Modifie les informations d'un champ.

## Source

```
BEGIN  
PERFORM login._token_assert (prm_token, FALSE, FALSE);  
PERFORM login._token_assert_interface (prm_token);  
UPDATE meta.info SET  
    int_id = prm_int_id,  
    inf_code = prm_code,  
    inf_libelle = prm_libelle,  
    inf_libelle_complet = prm_libelle_complet,  
    inf_etendu = prm_etendu,  
    inf_historique = prm_historique,  
    inf_multiple = prm_multiple  
WHERE inf_id = prm_inf_id;  
END;
```

## Name

meta\_info\_usage — Retourne les pages sur lesquelles est utilisé un champ.

## Synopsis

```
setof meta_info_usage meta_info_usage(prm_token, prm_inf_id);
```

```
int4 prm_token;  
int4 prm_inf_id;
```

## Description

Retourne les pages sur lesquelles est utilisé un champ.

## Source

```
DECLARE  
  row meta.meta_info_usage;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  FOR row IN  
    select cat_nom, por_libelle, ent_libelle, men_libelle, sme_libelle FROM meta.info  
      inner join meta.info_groupe using(inf_id)  
      inner join meta.groupe_infos using(gin_id)  
      inner join meta.sousmenu using(sme_id)  
      inner join meta.menu using(men_id)  
      inner join meta.entite using(ent_id)  
      inner join meta.portail using(por_id)  
      inner join meta.categorie using(cat_id)  
      where inf_id = prm_inf_id  
    LOOP  
      RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

meta\_infos\_contact\_update — Modifie les informations spécifiques d'un champ de type "contact".

## Synopsis

```
void    meta_infos_contact_update(prm_token,    prm_inf_id,    prm_filtre,  
prm_secteur);
```

```
int4    prm_token;  
int4    prm_inf_id;  
varchar prm_filtre;  
varchar prm_secteur;
```

## Description

Modifie les informations spécifiques d'un champ de type "contact".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info SET  
    inf__contact_filtre = prm_filtre,  
    inf__contact_secteur = prm_secteur  
  WHERE inf_id = prm_inf_id;  
END;
```

## Name

meta\_infos\_date\_update — Modifie les informations spécifiques d'un champ de type "date".

## Synopsis

```
void meta_infos_date_update(prm_token, prm_inf_id, prm__date_echeance,  
prm__date_echeance_icone, prm__date_echeance_secteur);
```

```
int4 prm_token;  
int4 prm_inf_id;  
bool prm__date_echeance;  
varchar prm__date_echeance_icone;  
varchar prm__date_echeance_secteur;
```

## Description

Modifie les informations spécifiques d'un champ de type "date".

## Source

```
BEGIN  
PERFORM login._token_assert (prm_token, FALSE, FALSE);  
PERFORM login._token_assert_interface (prm_token);  
UPDATE meta.info SET  
    inf__date_echeance = prm__date_echeance,  
    inf__date_echeance_icone = prm__date_echeance_icone,  
    inf__date_echeance_secteur = prm__date_echeance_secteur  
WHERE inf_id = prm_inf_id;  
END;
```

## Name

`meta_infos_etablissement_update` — Modifie les informations spécifiques d'un champ de type "etablissement".

## Synopsis

```
void meta_infos_etablissement_update(prm_token, prm_inf_id, prm_interne,  
prm_etablissement_secteur);
```

```
int4 prm_token;  
int4 prm_inf_id;  
bool prm_interne;  
varchar prm_etablissement_secteur;
```

## Description

Modifie les informations spécifiques d'un champ de type "etablissement".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info SET  
    inf_etablissement_interne = prm_interne,  
    inf_etablissement_secteur = prm_etablissement_secteur  
  WHERE inf_id = prm_inf_id;  
END;
```

## Name

meta\_infos\_groupe\_update — Modifie les informations spécifiques d'un champ de type "etablissement".

## Synopsis

```
void      meta_infos_groupe_update(prm_token,      prm_inf_id,      prm_type,  
prm_soustype);
```

```
int4 prm_token;  
int4 prm_inf_id;  
varchar prm_type;  
int4 prm_soustype;
```

## Description

Modifie les informations spécifiques d'un champ de type "etablissement".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info SET  
    inf_groupe_type = prm_type,  
    inf_groupe_soustype = prm_soustype  
  WHERE inf_id = prm_inf_id;  
END;
```



## Name

meta\_infos\_metier\_update — Modifie les informations spécifiques d'un champ de type "metier".

## Synopsis

```
void meta_infos_metier_update(prm_token, prm_inf_id, prm_secteur);
```

```
int4 prm_token;
```

```
int4 prm_inf_id;
```

```
varchar prm_secteur;
```

## Description

Modifie les informations spécifiques d'un champ de type "metier".

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.info SET
    inf__metier_secteur = prm_secteur
  WHERE inf_id = prm_inf_id;
END;
```

## Name

`meta_infos_selection_update` — Modifie les informations spécifiques d'un champ de type "selection".

## Synopsis

```
void meta_infos_selection_update(prm_token, prm_inf_id, prm_code);
```

```
int4 prm_token;  
int4 prm_inf_id;  
int4 prm_code;
```

## Description

Modifie les informations spécifiques d'un champ de type "selection".

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info SET  
    inf__selection_code = prm_code  
  WHERE inf_id = prm_inf_id;  
END;
```

## Name

`meta_infos_textelong_update` — Modifie les informations spécifiques d'un champ de type "textelong".

## Synopsis

```
void meta_infos_textelong_update(prm_token, prm_inf_id, prm_nblignes);
```

```
int4 prm_token;
```

```
int4 prm_inf_id;
```

```
int4 prm_nblignes;
```

## Description

Modifie les informations spécifiques d'un champ de type "textelong".

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.info SET
    inf__textelong_nblignes = prm_nblignes
  WHERE inf_id = prm_inf_id;
END;
```

## Name

meta\_infos\_type\_liste — Retourne la liste des types de champs.

## Synopsis

```
setof infos_type meta_infos_type_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des types de champs.

## Source

```
DECLARE
    row meta_infos_type%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta_infos_type
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`meta_ing_groupe_cycle_set` — Indique si un champ de type "groupe" utilise le cycle.

## Synopsis

```
void meta_ing_groupe_cycle_set(prm_token, prm_ing_id, prm_val);
```

```
int4 prm_token;  
int4 prm_ing_id;  
bool prm_val;
```

## Description

Indique si un champ de type "groupe" utilise le cycle.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info_groupe SET ing_groupe_cycle = prm_val WHERE ing_id = prm_ing_id;  
END;
```

## Name

`meta_ing_obligatoire_set` — Indique si un champ doit être rempli à la création d'une personne.

## Synopsis

```
void meta_ing_obligatoire_set(prm_token, prm_ing_id, prm_val);
```

```
int4 prm_token;  
int4 prm_ing_id;  
bool prm_val;
```

## Description

Indique si un champ doit être rempli à la création d'une personne.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.info_groupe SET ing_obligatoire = prm_val WHERE ing_id = prm_ing_id;  
END;
```

## Name

meta\_lien\_familial\_cherche — Recherche un lien familial par son nom.

## Synopsis

```
lien_familial meta_lien_familial_cherche(prm_token, prm_lfa_nom);
```

```
int4 prm_token;
```

```
varchar prm_lfa_nom;
```

## Description

Recherche un lien familial par son nom.

## Source

```
DECLARE
    ret meta.lien_familial;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM meta.lien_familial WHERE lfa_nom ilike prm_lfa_nom;
    RETURN ret;
END;
```

## Name

meta\_lien\_familial\_get — Retourne les informations d'un lien familial.

## Synopsis

```
lien_familial meta_lien_familial_get(prm_token, prm_lfa_id);
```

```
int4 prm_token;
```

```
int4 prm_lfa_id;
```

## Description

Retourne les informations d'un lien familial.

## Source

```
DECLARE
  ret meta.lien_familial;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.lien_familial WHERE lfa_id = prm_lfa_id;
  RETURN ret;
END;
```



## Name

meta\_lien\_familial\_liste — Retourne la liste des liens familiaux.

## Synopsis

```
setof lien_familial meta_lien_familial_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des liens familiaux.

## Source

```
DECLARE
    row meta.lien_familial;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.lien_familial ORDER BY lfa_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

meta\_menu\_add\_end — Ajoute une entrée de menu à un portail pour un type de personne.

## Synopsis

```
int4 meta_menu_add_end(prm_token, prm_por_id, prm_libelle, prm_ent_id);
```

```
int4 prm_token;  
int4 prm_por_id;  
varchar prm_libelle;  
int4 prm_ent_id;
```

## Description

Ajoute une entrée de menu à un portail pour un type de personne.

## Source

```
DECLARE  
    ret integer;  
    int_ordre integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    SELECT MAX(men_ordre) + 1 INTO int_ordre FROM meta.menu WHERE por_id = prm_por_id AND ent_id = prm_ent_id;  
    IF int_ordre ISNULL THEN int_ordre = 0; END IF;  
    INSERT INTO meta.menu(por_id, men_libelle, men_ordre, ent_id) VALUES (prm_por_id, prm_libelle, int_ordre, prm_ent_id);  
    RETURN ret;  
END;
```

## Name

meta\_menu\_delete — Supprime une entrée de menu.

## Synopsis

```
void meta_menu_delete(prm_token, prm_men_id);
```

```
int4 prm_token;
```

```
int4 prm_men_id;
```

## Description

Supprime une entrée de menu.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM meta.menu WHERE men_id = prm_men_id;
END;
```

## Name

meta\_menu\_deplacer\_bas — Déplace une entrée de menu vers le bas.

## Synopsis

```
void meta_menu_deplacer_bas(prm_token, prm_men_id);
```

```
int4 prm_token;
```

```
int4 prm_men_id;
```

## Description

Déplace une entrée de menu vers le bas.

## Source

```
BEGIN
PERFORM login._token_assert (prm_token, FALSE, FALSE);
PERFORM login._token_assert_interface (prm_token);
PERFORM meta.meta_menus_reordonne (prm_token,
    (SELECT por_id FROM meta.menu WHERE men_id = prm_men_id),
    (SELECT ent_id FROM meta.menu WHERE men_id = prm_men_id ));
IF (SELECT men_ordre FROM meta.menu WHERE men_id = prm_men_id) = (SELECT MAX(men_ordre) FROM meta.menu
    por_id = (SELECT por_id FROM meta.menu WHERE men_id = prm_men_id) AND
    ent_id = (SELECT ent_id FROM meta.menu WHERE men_id = prm_men_id)
THEN RETURN; END IF;
UPDATE meta.menu SET men_ordre = men_ordre + 1 WHERE men_id = prm_men_id;
UPDATE meta.menu SET men_ordre = men_ordre - 1 WHERE
    por_id = (SELECT por_id FROM meta.menu WHERE men_id = prm_men_id) AND
    ent_id = (SELECT ent_id FROM meta.menu WHERE men_id = prm_men_id) AND
    men_ordre = (SELECT men_ordre FROM meta.menu WHERE men_id = prm_men_id) AND
    men_id <> prm_men_id;
END;
```

## Name

meta\_menu\_deplacer\_haut — Déplace une entrée de menu vers le haut.

## Synopsis

```
void meta_menu_deplacer_haut(prm_token, prm_men_id);
```

```
int4 prm_token;  
int4 prm_men_id;
```

## Description

Déplace une entrée de menu vers le haut.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  PERFORM meta.meta_menus_reordonne (prm_token,  
    (SELECT por_id FROM meta.menu WHERE men_id = prm_men_id),  
    (SELECT ent_id FROM meta.menu WHERE men_id = prm_men_id ) );  
  IF (SELECT men_ordre FROM meta.menu WHERE men_id = prm_men_id) = 1 THEN RETURN; END IF;  
  UPDATE meta.menu SET men_ordre = men_ordre - 1 WHERE men_id = prm_men_id;  
  UPDATE meta.menu SET men_ordre = men_ordre + 1 WHERE  
    por_id = (SELECT por_id FROM meta.menu WHERE men_id = prm_men_id) AND  
    ent_id = (SELECT ent_id FROM meta.menu WHERE men_id = prm_men_id) AND  
    men_ordre = (SELECT men_ordre FROM meta.menu WHERE men_id = prm_men_id) AND  
    men_id <> prm_men_id;  
END;
```

## Name

meta\_menu\_infos — Retourne les informations d'une entrée de menu de fiche personne.

## Synopsis

```
menu meta_menu_infos(prm_token, prm_men_id);
```

```
int4 prm_token;
```

```
int4 prm_men_id;
```

## Description

Retourne les informations d'une entrée de menu de fiche personne.

## Source

```
DECLARE
    ret meta.menu%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM meta.menu WHERE men_id = prm_men_id;
    RETURN ret;
END;
```

## Name

meta\_menu\_liste — Retourne la liste des menus pour un portail et un type de personne donnés.

## Synopsis

```
setof menu meta_menu_liste(prm_token, prm_por_id, prm_ent_code);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

```
varchar prm_ent_code;
```

## Description

Retourne la liste des menus pour un portail et un type de personne donnés.

## Source

```
DECLARE
    row meta.menu%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.menu WHERE por_id = prm_por_id AND ent_id = (SELECT ent_id FROM meta.entite WHERE
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

meta\_menu\_rename — Renomme une entrée de menu.

## Synopsis

```
void meta_menu_rename(prm_token, prm_men_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_men_id;
```

```
varchar prm_libelle;
```

## Description

Renomme une entrée de menu.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.menu SET men_libelle = prm_libelle WHERE men_id = prm_men_id;
END;
```



## Name

`meta_menu_un_seul_niveau` — Retourne TRUE si le menu d'une fiche personne est à un seul niveau.

## Synopsis

```
bool meta_menu_un_seul_niveau(prm_token, prm_por_id, prm_ent_code);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

```
varchar prm_ent_code;
```

## Description

Retourne TRUE si le menu d'une fiche personne est à un seul niveau.

## Source

```
DECLARE
  ret boolean;
  mens meta.menu;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR mens IN
    SELECT men_id FROM meta.menu WHERE por_id = prm_por_id AND ent_id = (SELECT ent_id FROM meta.entite W
  LOOP
    IF (SELECT COUNT(*) FROM meta.sousmenu WHERE men_id = mens.men_id) > 1 THEN
      RETURN FALSE;
    END IF;
  END LOOP;
  RETURN TRUE;
END;
```

## Name

meta\_menus\_reordonne — Réordonne les entrées de menu d'un portail pour un type de personne.

## Synopsis

```
void meta_menus_reordonne(prm_token, prm_por_id, prm_ent_id);
```

```
int4 prm_token;  
int4 prm_por_id;  
int4 prm_ent_id;
```

## Description

Réordonne les entrées de menu d'un portail pour un type de personne.

## Source

```
DECLARE  
  i integer;  
  row RECORD;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  i = 1;  
  FOR row IN  
    SELECT men_id FROM meta.menu WHERE por_id = prm_por_id AND ent_id = prm_ent_id ORDER BY men_ordre  
  LOOP  
    UPDATE meta.menu SET men_ordre = i WHERE men_id = row.men_id;  
    i = i + 1;  
  END LOOP;  
END;
```

## Name

meta\_menus\_supprime\_recuratif — Supprime un menu de fiche personne et toutes ses fiches.

## Synopsis

```
void meta_menus_supprime_recuratif(prm_token, prm_ent_code, prm_por_id);
```

```
int4 prm_token;
```

```
varchar prm_ent_code;
```

```
int4 prm_por_id;
```

## Description

Supprime un menu de fiche personne et toutes ses fiches.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM meta.info_groupe WHERE gin_id IN (SELECT gin_id FROM meta.groupe_infos WHERE sme_id IN (SEL
  DELETE FROM meta.groupe_infos WHERE sme_id IN (SELECT sme_id FROM meta.sousmenu WHERE men_id IN (SELECT
  DELETE FROM meta.sousmenu WHERE men_id IN (SELECT men_id FROM meta.menu WHERE ent_id = (SELECT ent_id F
  DELETE FROM meta.menu WHERE ent_id = (SELECT ent_id FROM meta.entite WHERE ent_code = prm_ent_code) AND
END;
```

## Name

meta\_portail\_add — Ajoute un portail.

## Synopsis

```
int4 meta_portail_add(prm_token, prm_cat_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_cat_id;
```

```
varchar prm_libelle;
```

## Description

Ajoute un portail.

## Source

```
DECLARE
    ret integer;
    row meta.entite;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    INSERT INTO meta.portail (cat_id, por_libelle) VALUES (prm_cat_id, prm_libelle) RETURNING por_id INTO r
    FOR row IN SELECT * FROM meta.entite LOOP
        INSERT INTO permission.droit_ajout_entite_portail (ent_code, por_id, daj_droit) VALUES (row.ent_code,
    END LOOP;
    RETURN ret;
END;
```

## Name

meta\_portail\_delete — Supprime un portail.

## Synopsis

```
void meta_portail_delete(prm_token, prm_por_id);
```

```
int4 prm_token;  
int4 prm_por_id;
```

## Description

Supprime un portail.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  DELETE FROM permission.droit_ajout_entite_portail WHERE por_id = prm_por_id;  
  DELETE FROM meta.portail WHERE por_id = prm_por_id;  
END;
```

## Name

meta\_portail\_delete\_rec — Supprime un portail et tout ce qu'il contient.

## Synopsis

```
void meta_portail_delete_rec(prm_token, prm_por_id);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

## Description

Supprime un portail et tout ce qu'il contient.

## Source

```
DECLARE
    row_ent meta.entite;
    row_tom meta.topmenu;
    row_tsm meta.topsousmenu;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row_ent IN SELECT * FROM meta.entite LOOP
        PERFORM meta.meta_menus_supprime_recurcif (prm_token, row_ent.ent_code, prm_por_id);
    END LOOP;
    FOR row_tom IN SELECT * FROM meta.topmenu WHERE por_id = prm_por_id LOOP
        FOR row_tsm IN SELECT * FROM meta.topsousmenu WHERE tom_id = row_tom.tom_id LOOP
            PERFORM meta.meta_topsousmenu_delete (prm_token, row_tsm.tsm_id);
        END LOOP;
        PERFORM meta.meta_topmenu_delete (prm_token, row_tom.tom_id);
    END LOOP;
    DELETE FROM permission.droit_ajout_entite_portail WHERE por_id = prm_por_id;
    DELETE FROM notes.theme_portail WHERE por_id = prm_por_id;
    DELETE FROM meta.portail WHERE por_id = prm_por_id;
END;
```

## Name

meta\_portail\_get — Retourne les informations sur un portail.

## Synopsis

```
portail meta_portail_get(prm_token, prm_cat_id, prm_por_id);
```

```
int4 prm_token;  
int4 prm_cat_id;  
int4 prm_por_id;
```

## Description

Retourne les informations sur un portail.

## Source

```
DECLARE  
    ret meta.portail;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM meta.portail WHERE cat_id = prm_cat_id AND por_id = prm_por_id;  
    RETURN ret;  
END;
```

## Name

meta\_portail\_infos — Retourne les informations sur un portail.

## Synopsis

```
portail meta_portail_infos(prm_token, prm_por_id);
```

```
int4 prm_token;  
int4 prm_por_id;
```

## Description

Retourne les informations sur un portail.

## Source

```
DECLARE  
    ret meta.portail;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM meta.portail WHERE por_id = prm_por_id;  
    RETURN ret;  
END;
```



## Name

meta\_portail\_liste — Retourne la liste des portails définis pour une catégorie donnée.

## Synopsis

```
setof portail meta_portail_liste(prm_token, prm_cat_id);
```

```
int4 prm_token;  
int4 prm_cat_id;
```

## Description

Retourne la liste des portails définis pour une catégorie donnée.

## Source

```
DECLARE  
  row meta.portail;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT * FROM meta.portail WHERE (prm_cat_id ISNULL OR cat_id = prm_cat_id) ORDER BY por_libelle  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

meta\_portail\_purge — Vide un portail de ses menus.

## Synopsis

```
void meta_portail_purge(prm_token, prm_por_id);
```

```
int4 prm_token;  
int4 prm_por_id;
```

## Description

Vide un portail de ses menus.

## Source

```
DECLARE  
  row_ent meta.entite;  
  row_tom meta.topmenu;  
  row_tsm meta.topsousmenu;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  FOR row_ent IN SELECT * FROM meta.entite LOOP  
    PERFORM meta.meta_menus_supprime_rekursif (prm_token, row_ent.ent_code, prm_por_id);  
  END LOOP;  
  FOR row_tom IN SELECT * FROM meta.topmenu WHERE por_id = prm_por_id LOOP  
    FOR row_tsm IN SELECT * FROM meta.topsousmenu WHERE tom_id = row_tom.tom_id LOOP  
      PERFORM meta.meta_topsousmenu_delete (prm_token, row_tsm.tsm_id);  
    END LOOP;  
    PERFORM meta.meta_topmenu_delete (prm_token, row_tom.tom_id);  
  END LOOP;  
END;
```

## Name

meta\_portail\_rename — Renomme un portail.

## Synopsis

```
void meta_portail_rename(prm_token, prm_por_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

```
varchar prm_libelle;
```

## Description

Renomme un portail.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.portail SET por_libelle = prm_libelle WHERE por_id = prm_por_id;
END;
```

## Name

`meta_secteur_get` — Retourne les informations sur un secteur.

## Synopsis

```
secteur meta_secteur_get(prm_token, prm_sec_id);
```

```
int4 prm_token;  
int4 prm_sec_id;
```

## Description

Retourne les informations sur un secteur.

## Source

```
DECLARE  
    ret meta.secteur;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM meta.secteur WHERE sec_id = prm_sec_id;  
    RETURN ret;  
END;
```

## Name

`meta_secteur_get_par_code` — Retourne les informations sur un secteur à partir de son code.

## Synopsis

```
secteur meta_secteur_get_par_code(prm_token, prm_sec_code);
```

```
int4 prm_token;
```

```
varchar prm_sec_code;
```

## Description

Retourne les informations sur un secteur à partir de son code.

## Source

```
DECLARE
    ret meta.secteur;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM meta.secteur WHERE sec_code = prm_sec_code;
    RETURN ret;
END;
```

## Name

`meta_secteur_liste` — Retourne la liste des secteurs.

## Synopsis

```
setof secteur meta_secteur_liste(prm_token, prm_est_prise_en_charge);
```

```
int4 prm_token;
```

```
bool prm_est_prise_en_charge;
```

## Description

Retourne la liste des secteurs.

## Source

```
DECLARE
    row meta.secteur;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.secteur WHERE (prm_est_prise_en_charge ISNULL OR sec_est_prise_en_charge = prm_est.
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`meta_secteur_save` — Modifie les informations d'un secteur.

## Synopsis

```
void meta_secteur_save(prm_token, prm_code, prm_nom, prm_icone);
```

```
int4 prm_token;  
varchar prm_code;  
varchar prm_nom;  
varchar prm_icone;
```

## Description

Modifie les informations d'un secteur.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE meta.secteur SET sec_nom = prm_nom, sec_icone = prm_icone WHERE sec_code = prm_code;  
END;
```

## Name

meta\_secteur\_type\_add — Ajoute un type à un secteur.

## Synopsis

```
int4 meta_secteur_type_add(prm_token, prm_sec_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_sec_id;  
varchar prm_nom;
```

## Description

Ajoute un type à un secteur.

## Source

```
DECLARE  
  ret INTEGER;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, TRUE);  
  INSERT INTO meta.secteur_type (sec_id, set_nom) VALUES (prm_sec_id, prm_nom) RETURNING set_id INTO ret;  
  RETURN ret;  
END;
```



## Name

`meta_secteur_type_delete` — Supprime un type d'un secteur.

## Synopsis

```
void meta_secteur_type_delete(prm_token, prm_set_id);
```

```
int4 prm_token;
```

```
int4 prm_set_id;
```

## Description

Supprime un type d'un secteur.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, TRUE);
  DELETE FROM meta.secteur_type WHERE set_id = prm_set_id;
END;
```

## Name

meta\_secteur\_type\_liste — Retourne la liste des types d'un secteur.

## Synopsis

```
setof secteur_type meta_secteur_type_liste(prm_token, prm_sec_id);
```

```
int4 prm_token;
```

```
int4 prm_sec_id;
```

## Description

Retourne la liste des types d'un secteur.

## Source

```
DECLARE
    row meta.secteur_type;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.secteur_type WHERE sec_id = prm_sec_id ORDER BY set_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`meta_secteur_type_liste_par_code` — Retourne la liste des types d'un secteur, à partir du code du secteur.

## Synopsis

```
setof          secteur_type          meta_secteur_type_liste_par_code(prm_token,  
prm_sec_code);
```

```
int4 prm_token;  
varchar prm_sec_code;
```

## Description

Retourne la liste des types d'un secteur, à partir du code du secteur.

## Source

```
DECLARE  
  row meta.secteur_type;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT meta.secteur_type.* FROM meta.secteur_type  
      INNER JOIN meta.secteur USING(sec_id)  
      WHERE sec_code = prm_sec_code ORDER BY set_nom  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

meta\_secteur\_type\_update — Modifie les informations d'un type d'un secteur.

## Synopsis

```
void meta_secteur_type_update(prm_token, prm_set_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_set_id;  
varchar prm_nom;
```

## Description

Modifie les informations d'un type d'un secteur.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, TRUE);  
  UPDATE meta.secteur_type SET set_nom = prm_nom WHERE set_id = prm_set_id;  
END;
```

## Name

meta\_selection\_add\_avec\_id — Ajoute une liste de sélection (utilisé avec la banque de champs).

## Synopsis

```
int4 meta_selection_add_avec_id(prm_token, prm_id, prm_code, prm_libelle,  
prm_info);
```

```
int4 prm_token;  
int4 prm_id;  
varchar prm_code;  
varchar prm_libelle;  
varchar prm_info;
```

## Description

Ajoute une liste de sélection (utilisé avec la banque de champs).

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  INSERT INTO meta.selection (sel_id, sel_code, sel_libelle, sel_info) VALUES (prm_id, prm_code, prm_libelle, prm_info)  
    RETURNING sel_id INTO ret;  
  PERFORM setval ('meta.selection_sel_id_seq', coalesce((select max(sel_id)+1 from meta.selection), 1), f);  
  RETURN ret;  
END;
```

## Name

`meta_selection_dernier` — Retourne la dernière liste de sélection en base (utilisé avec la banque de champs).

## Synopsis

```
int4 meta_selection_dernier(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la dernière liste de sélection en base (utilisé avec la banque de champs).

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT MAX(sel_id) INTO ret FROM meta.selection;
  RETURN ret;
END;
```

## Name

meta\_selection\_entree\_add — Ajoute une entrée à une liste de sélection.

## Synopsis

```
int4    meta_selection_entree_add(prm_token,    prm_sel_id,    prm_libelle,  
prm_ordre);
```

```
int4 prm_token;  
int4 prm_sel_id;  
varchar prm_libelle;  
int4 prm_ordre;
```

## Description

Ajoute une entrée à une liste de sélection.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    INSERT INTO meta.selection_entree (sel_id, sen_libelle, sen_ordre) VALUES (prm_sel_id, prm_libelle, prm  
        RETURNING sen_id INTO ret;  
    RETURN ret;  
END;
```

## Name

`meta_selection_entree_get` — Retourne les informations sur une entrée de liste de sélection.

## Synopsis

```
selection_entree meta_selection_entree_get(prm_token, prm_sen_id);
```

```
int4 prm_token;
```

```
int4 prm_sen_id;
```

## Description

Retourne les informations sur une entrée de liste de sélection.

## Source

```
DECLARE
  ret meta.selection_entree;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.selection_entree WHERE sen_id = prm_sen_id;
  RETURN ret;
END;
```



## Name

`meta_selection_entree_liste` — Retourne les entrées d'une liste de sélection.

## Synopsis

```
setof selection_entree meta_selection_entree_liste(prm_token, prm_sel_id);
```

```
int4 prm_token;  
int4 prm_sel_id;
```

## Description

Retourne les entrées d'une liste de sélection.

## Source

```
DECLARE  
  row meta.selection_entree%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT * FROM meta.selection_entree WHERE sel_id = prm_sel_id ORDER BY sen_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`meta_selection_entree_liste_par_cha` — Retourne les entrées d'une liste de sélection d'après l'identifiant du champ de type sélection.

## Synopsis

```
setof selection_entree meta_selection_entree_liste_par_cha(prm_token,  
prm_cha_id);
```

```
int4 prm_token;  
int4 prm_cha_id;
```

## Description

Retourne les entrées d'une liste de sélection d'après l'identifiant du champ de type sélection.

## Source

```
DECLARE  
  row meta.selection_entree%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT selection_entree.* FROM meta.selection_entree  
    INNER JOIN meta.info ON info.inf__selection_code = sel_id  
    INNER JOIN liste.champ USING (inf_id)  
    WHERE cha_id = prm_cha_id ORDER BY sen_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`meta_selection_entree_liste_par_code` — Retourne les entrées d'une liste de sélection à partir du code de la liste.

## Synopsis

```
setof selection_entree meta_selection_entree_liste_par_code(prm_token,  
prm_sel_code);
```

```
int4 prm_token;  
varchar prm_sel_code;
```

## Description

Retourne les entrées d'une liste de sélection à partir du code de la liste.

## Source

```
DECLARE  
  row meta.selection_entree%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT * FROM meta.selection_entree WHERE sel_id = (SELECT sel_id FROM meta.selection WHERE sel_code  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

meta\_selection\_entree\_set\_ordre — Modifie l'ordre d'apparition d'une entrée dans la liste de sélection.

## Synopsis

```
void meta_selection_entree_set_ordre(prm_token, prm_sen_id, prm_ordre);
```

```
int4 prm_token;  
int4 prm_sen_id;  
int4 prm_ordre;
```

## Description

Modifie l'ordre d'apparition d'une entrée dans la liste de sélection.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    UPDATE meta.selection_entree SET sen_ordre = prm_ordre WHERE sen_id = prm_sen_id;  
END;
```

## Name

`meta_selection_entree_supprime` — Supprime une entrée de liste de sélection.

## Synopsis

```
void meta_selection_entree_supprime(prm_token, prm_sen_id);
```

```
int4 prm_token;
```

```
int4 prm_sen_id;
```

## Description

Supprime une entrée de liste de sélection.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM meta.selection_entree WHERE sen_id = prm_sen_id;
END;
```

## Name

meta\_selection\_infos — Retourne les infos d'une liste de sélection.

## Synopsis

```
selection meta_selection_infos(prm_token, prm_sel_id);
```

```
int4 prm_token;  
int4 prm_sel_id;
```

## Description

Retourne les infos d'une liste de sélection.

## Source

```
DECLARE  
  ret meta.selection%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  SELECT * INTO ret FROM meta.selection WHERE sel_id = prm_sel_id;  
  RETURN ret;  
END;
```

## Name

`meta_selection_infos_par_code` — Retourne les infos d'une liste de sélection à partir de son code.

## Synopsis

```
selection meta_selection_infos_par_code(prm_token, prm_sel_code);
```

```
int4 prm_token;
```

```
varchar prm_sel_code;
```

## Description

Retourne les infos d'une liste de sélection à partir de son code.

## Source

```
DECLARE
  ret meta.selection%ROWTYPE;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.selection WHERE sel_code = prm_sel_code;
  RETURN ret;
END;
```

## Name

`meta_selection_liste` — Retourne la liste des listes de sélection.

## Synopsis

```
setof selection meta_selection_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des listes de sélection.

## Source

```
DECLARE
  row meta.selection;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  FOR row IN
    SELECT * FROM meta.selection ORDER BY sel_libelle
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```



## Name

meta\_selection\_update — Modifie les informations d'une liste de sélection.

## Synopsis

```
void meta_selection_update(prm_token, prm_sel_id, prm_code, prm_libelle,  
prm_info);
```

```
int4 prm_token;  
int4 prm_sel_id;  
varchar prm_code;  
varchar prm_libelle;  
varchar prm_info;
```

## Description

Modifie les informations d'une liste de sélection.

## Source

```
BEGIN  
  UPDATE meta.selection SET  
    sel_code = prm_code,  
    sel_libelle = prm_libelle,  
    sel_info = prm_info  
  WHERE sel_id = prm_sel_id;  
END;
```

## Name

meta\_sousmenu\_add\_end — Ajoute une fiche à un menu personne.

## Synopsis

```
int4 meta_sousmenu_add_end(prm_token, prm_men_id, prm_libelle, prm_type,  
prm_type_id, prm_droit_modif, prm_droit_suppression, prm_icone);
```

```
int4 prm_token;  
int4 prm_men_id;  
varchar prm_libelle;  
varchar prm_type;  
int4 prm_type_id;  
bool prm_droit_modif;  
bool prm_droit_suppression;  
varchar prm_icone;
```

## Description

Ajoute une fiche à un menu personne.

## Source

```
DECLARE  
  ret integer;  
  int_ordre integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  SELECT MAX(sme_ordre) + 1 INTO int_ordre FROM meta.sousmenu WHERE men_id = prm_men_id;  
  IF int_ordre ISNULL THEN int_ordre = 0; END IF;  
  INSERT INTO meta.sousmenu(men_id, sme_libelle, sme_ordre, sme_type, sme_type_id, sme_droit_modif, sme_d  
  RETURN ret;  
END;
```

## Name

meta\_sousmenu\_delete — Supprime une fiche d'un menu personne.

## Synopsis

```
void meta_sousmenu_delete(prm_token, prm_sme_id);
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

## Description

Supprime une fiche d'un menu personne.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM procedure.procedure_affectation WHERE sme_id = prm_sme_id;
  DELETE FROM meta.sousmenu WHERE sme_id = prm_sme_id;
END;
```

## Name

meta\_sousmenu\_deplacer\_bas — Déplace vers le bas une fiche d'un menu personne.

## Synopsis

```
void meta_sousmenu_deplacer_bas(prm_token, prm_sme_id);
```

```
int4 prm_token;  
int4 prm_sme_id;
```

## Description

Déplace vers le bas une fiche d'un menu personne.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  PERFORM meta.meta_sousmenus_reordonne (prm_token, (SELECT men_id FROM meta.sousmenu WHERE sme_id = prm_  
  IF (SELECT sme_ordre FROM meta.sousmenu WHERE sme_id = prm_sme_id) = (SELECT MAX(sme_ordre) FROM meta.s  
      men_id = (SELECT men_id FROM meta.sousmenu WHERE sme_id = prm_sme_id))  
  THEN RETURN; END IF;  
  UPDATE meta.sousmenu SET sme_ordre = sme_ordre + 1 WHERE sme_id = prm_sme_id;  
  UPDATE meta.sousmenu SET sme_ordre = sme_ordre - 1 WHERE  
    men_id = (SELECT men_id FROM meta.sousmenu WHERE sme_id = prm_sme_id) AND  
    sme_ordre = (SELECT sme_ordre FROM meta.sousmenu WHERE sme_id = prm_sme_id) AND  
    sme_id <> prm_sme_id;  
END;
```

## Name

meta\_sousmenu\_deplacer\_haut — Déplace vers le haut une fiche d'un menu personne.

## Synopsis

```
void meta_sousmenu_deplacer_haut(prm_token, prm_sme_id);
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

## Description

Déplace vers le haut une fiche d'un menu personne.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  PERFORM meta.meta_sousmenus_reordonne (prm_token, (SELECT men_id FROM meta.sousmenu WHERE sme_id = prm_
  IF (SELECT sme_ordre FROM meta.sousmenu WHERE sme_id = prm_sme_id) = 1 THEN RETURN; END IF;
  UPDATE meta.sousmenu SET sme_ordre = sme_ordre - 1 WHERE sme_id = prm_sme_id;
  UPDATE meta.sousmenu SET sme_ordre = sme_ordre + 1 WHERE
    men_id = (SELECT men_id FROM meta.sousmenu WHERE sme_id = prm_sme_id) AND
    sme_ordre = (SELECT sme_ordre FROM meta.sousmenu WHERE sme_id = prm_sme_id) AND
    sme_id <> prm_sme_id;
END;
```

## Name

meta\_sousmenu\_infos — Retourne les informations sur une fiche d'un menu personne.

## Synopsis

```
sousmenu meta_sousmenu_infos(prm_token, prm_sme_id);
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

## Description

Retourne les informations sur une fiche d'un menu personne.

## Source

```
DECLARE
    ret meta.sousmenu%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM meta.sousmenu WHERE sme_id = prm_sme_id;
    RETURN ret;
END;
```

## Name

meta\_sousmenu\_liste — Retourne la liste des fiches d'un menu personne.

## Synopsis

```
setof sousmenu meta_sousmenu_liste(prm_token, prm_men_id);
```

```
int4 prm_token;
```

```
int4 prm_men_id;
```

## Description

Retourne la liste des fiches d'un menu personne.

## Source

```
DECLARE
    row meta.sousmenu%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.sousmenu WHERE men_id = prm_men_id ORDER BY sme_ordre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

meta\_sousmenu\_rename — Renomme une fiche de menu personne.

## Synopsis

```
void meta_sousmenu_rename(prm_token, prm_sme_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_sme_id;
```

```
varchar prm_libelle;
```

## Description

Renomme une fiche de menu personne.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.sousmenu SET sme_libelle = prm_libelle WHERE sme_id = prm_sme_id;
END;
```



## Name

`meta_sousmenu_set_droit_modif` — Indique si l'utilisateur a droit de modification sur cette fiche.

## Synopsis

```
void meta_sousmenu_set_droit_modif(prm_token, prm_id, prm_droit_modif);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

```
bool prm_droit_modif;
```

## Description

Indique si l'utilisateur a droit de modification sur cette fiche.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  UPDATE meta.sousmenu SET sme_droit_modif = prm_droit_modif WHERE sme_id = prm_id;
END;
```

## Name

`meta_sousmenu_set_droit_suppression` — Indique si l'utilisateur a droit de suppression sur cette fiche.

## Synopsis

```
void      meta_sousmenu_set_droit_suppression(prm_token,      prm_id,  
prm_droit_suppression);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

```
bool prm_droit_suppression;
```

## Description

Indique si l'utilisateur a droit de suppression sur cette fiche.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE meta.sousmenu SET sme_droit_suppression = prm_droit_suppression WHERE sme_id = prm_id;  
END;
```

## Name

meta\_sousmenu\_set\_type — Modifie le type d'une fiche de menu personne.

## Synopsis

```
void meta_sousmenu_set_type(prm_token, prm_sme_id, prm_type, prm_type_id,  
prm_icone);
```

```
int4 prm_token;  
int4 prm_sme_id;  
varchar prm_type;  
int4 prm_type_id;  
varchar prm_icone;
```

## Description

Modifie le type d'une fiche de menu personne.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.sousmenu SET sme_type = prm_type, sme_type_id = prm_type_id, sme_icone = prm_icone WHERE sme_id = prm_sme_id;  
END;
```

## Name

`meta_sousmenus_liste_depuis_topmenu` — Retourne la liste des fiches d'un type de personne accessible dans un portail.

## Synopsis

```
setof                                meta_sousmenus_liste_depuis_topmenu  
meta_sousmenus_liste_depuis_topmenu(prm_token, prm_tom_id, prm_ent_code);
```

```
int4 prm_token;  
int4 prm_tom_id;  
varchar prm_ent_code;
```

## Description

Retourne la liste des fiches d'un type de personne accessible dans un portail.

## Source

```
DECLARE  
  row meta.meta_sousmenus_liste_depuis_topmenu;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  FOR row IN SELECT men_id, menu.men_libelle, sousmenu.sme_id, sousmenu.sme_libelle FROM meta.sousmenu  
    INNER JOIN meta.menu USING(men_id)  
    WHERE ent_id = (SELECT ent_id FROM meta.entite WHERE ent_code = prm_ent_code)  
      AND por_id = (SELECT por_id FROM meta.topmenu WHERE tom_id = prm_tom_id)  
    ORDER BY men_ordre, sme_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

meta\_sousmenus\_reordonne — Réordonne des fiches m'un menu personne.

## Synopsis

```
void meta_sousmenus_reordonne(prm_token, prm_men_id);
```

```
int4 prm_token;
```

```
int4 prm_men_id;
```

## Description

Réordonne des fiches m'un menu personne.

## Source

```
DECLARE
  i integer;
  row RECORD;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  i = 1;
  FOR row IN
    SELECT sme_id FROM meta.sousmenu WHERE men_id = prm_men_id ORDER BY sme_ordre
  LOOP
    UPDATE meta.sousmenu SET sme_ordre = i WHERE sme_id = row.sme_id;
    i = i + 1;
  END LOOP;
END;
```

## Name

meta\_statut\_usager\_calcule

## Synopsis

```
int4 meta_statut_usager_calcule(prm_inf_code, prm_per_id);
```

```
varchar prm_inf_code;
```

```
int4 prm_per_id;
```

## Source

```

DECLARE
    ret integer;
    pin integer;
    pii integer;
BEGIN
    SELECT pin_id INTO pin FROM personne_info WHERE inf_code = prm_inf_code AND per_id = prm_per_id;
    IF NOT FOUND THEN
        INSERT INTO personne_info (inf_code, per_id) VALUES (prm_inf_code, prm_per_id) RETURNING pin_id INTO pin;
    END IF;

    SELECT pii_id INTO pii FROM personne_info_integer WHERE pin_id = pin;
    IF NOT FOUND THEN
        INSERT INTO personne_info_integer (pin_id) VALUES (pin) RETURNING pii_id INTO pii;
    END IF;

    ret = 5;
    IF EXISTS (SELECT 1 FROM personne_groupe WHERE per_id = prm_per_id AND peg_cycle_statut = 3) THEN -- Terminé
        ret = 1; -- Sorti
    END IF;
    IF EXISTS (SELECT 1 FROM personne_groupe WHERE per_id = prm_per_id AND peg_cycle_statut = 1) THEN -- Début
        ret = 2; -- Pré-admission
    END IF;
    IF EXISTS (SELECT 1 FROM personne_groupe WHERE per_id = prm_per_id AND peg_cycle_statut = 2 AND (peg_debut = 0 OR peg_debut = 1)) THEN
        ret = 3; -- Admission
    END IF;
    IF EXISTS (SELECT 1 FROM personne_groupe WHERE per_id = prm_per_id AND peg_cycle_statut = 2 AND (peg_debut = 2 OR peg_debut = 3)) THEN
        ret = 4; -- Présent
    END IF;

    UPDATE personne_info_integer SET pii_valeur = ret WHERE pii_id = pii;
    -- RAISE WARNING 'statut % : %', prm_per_id, ret;
    RETURN ret;
END;

```

## Name

meta\_topmenu\_add\_end — Ajoute une entrée dans un menu principal.

## Synopsis

```
int4 meta_topmenu_add_end(prm_token, prm_por_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

```
varchar prm_libelle;
```

## Description

Ajoute une entrée dans un menu principal.

## Source

```
DECLARE
  ret integer;
  int_ordre integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  SELECT MAX(tom_ordre) + 1 INTO int_ordre FROM meta.topmenu WHERE por_id = prm_por_id;
  IF int_ordre ISNULL THEN int_ordre = 0; END IF;
  INSERT INTO meta.topmenu (por_id, tom_libelle, tom_ordre) VALUES (prm_por_id, prm_libelle, int_ordre)
  RETURNING tom_id INTO ret;
  RETURN ret;
END;
```

## Name

meta\_topmenu\_delete — Supprime une entrée dans un menu principal.

## Synopsis

```
void meta_topmenu_delete(prm_token, prm_tom_id);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

## Description

Supprime une entrée dans un menu principal.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM meta.topmenu WHERE tom_id = prm_tom_id;
END;
```



## Name

meta\_topmenu\_deplacer\_bas — Déplace une entrée de menu principal vers le bas.

## Synopsis

```
void meta_topmenu_deplacer_bas(prm_token, prm_tom_id);
```

```
int4 prm_token;  
int4 prm_tom_id;
```

## Description

Déplace une entrée de menu principal vers le bas.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  PERFORM meta.meta_topmenus_reordonne (prm_token, (SELECT por_id FROM meta.topmenu WHERE tom_id = prm_tom_id));  
  IF (SELECT tom_ordre FROM meta.topmenu WHERE tom_id = prm_tom_id) = (SELECT MAX(tom_ordre) FROM meta.topmenu WHERE tom_id = prm_tom_id)  
    THEN RETURN; END IF;  
  UPDATE meta.topmenu SET tom_ordre = tom_ordre + 1 WHERE tom_id = prm_tom_id;  
  UPDATE meta.topmenu SET tom_ordre = tom_ordre - 1 WHERE  
    por_id = (SELECT por_id FROM meta.topmenu WHERE tom_id = prm_tom_id) AND  
    tom_ordre = (SELECT tom_ordre FROM meta.topmenu WHERE tom_id = prm_tom_id) AND  
    tom_id <> prm_tom_id;  
END;
```

## Name

meta\_topmenu\_deplacer\_haut — Déplace une entrée de menu principal vers le haut.

## Synopsis

```
void meta_topmenu_deplacer_haut(prm_token, prm_tom_id);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

## Description

Déplace une entrée de menu principal vers le haut.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  PERFORM meta.meta_topmenus_reordonne (prm_token, (SELECT por_id FROM meta.topmenu WHERE tom_id = prm_tom_id));
  IF (SELECT tom_ordre FROM meta.topmenu WHERE tom_id = prm_tom_id) = 1 THEN RETURN; END IF;
  UPDATE meta.topmenu SET tom_ordre = tom_ordre - 1 WHERE tom_id = prm_tom_id;
  UPDATE meta.topmenu SET tom_ordre = tom_ordre + 1 WHERE
    por_id = (SELECT por_id FROM meta.topmenu WHERE tom_id = prm_tom_id) AND
    tom_ordre = (SELECT tom_ordre FROM meta.topmenu WHERE tom_id = prm_tom_id) AND
    tom_id <> prm_tom_id;
END;
```

## Name

meta\_topmenu\_events — Retourne la liste des entrées de menu contenant des vues événements (pour webdav).

## Synopsis

```
setof topmenu meta_topmenu_events(prm_token, prm_por_id);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

## Description

Retourne la liste des entrées de menu contenant des vues événements (pour webdav).

## Source

```
DECLARE
    row meta.topmenu%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT topmenu.* FROM meta.topmenu
            INNER JOIN meta.topsousmenu USING(tom_id)
            WHERE tsm_type = 'event' AND por_id = prm_por_id ORDER BY tom_ordre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`meta_topmenu_get` — Retourne les informations d'une entrée de menu principal (pour webdav).

## Synopsis

```
topmenu meta_topmenu_get(prm_token, prm_tom_id);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

## Description

Retourne les informations d'une entrée de menu principal (pour webdav).

## Source

```
DECLARE
  ret meta.topmenu;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.topmenu WHERE tom_id = prm_tom_id;
  RETURN ret;
END;
```

## Name

meta\_topmenu\_liste — Retourne la liste des entrées de menu principal d'un portail donné.

## Synopsis

```
setof topmenu meta_topmenu_liste(prm_token, prm_por_id);
```

```
int4 prm_token;
```

```
int4 prm_por_id;
```

## Description

Retourne la liste des entrées de menu principal d'un portail donné.

## Source

```
DECLARE
    row meta.topmenu%ROWTYPE;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT * FROM meta.topmenu WHERE por_id = prm_por_id ORDER BY tom_ordre
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`meta_topmenu_liste_contenant_type` — Retourne la liste des entrées de menu contenant des fiches de type liste pour une catégorie de personne donnée (pour webdav).

## Synopsis

```
setof topmenu meta_topmenu_liste_contenant_type(prm_token, prm_por_id,  
prm_type);
```

```
int4 prm_token;  
int4 prm_por_id;  
varchar prm_type;
```

## Description

Retourne la liste des entrées de menu contenant des fiches de type liste pour une catégorie de personne donnée (pour webdav).

## Source

```
DECLARE  
  row meta.topmenu%ROWTYPE;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT topmenu.* FROM meta.topmenu  
      INNER JOIN meta.topsousmenu USING(tom_id)  
    INNER JOIN liste.liste ON liste.lis_id = tsm_type_id  
    INNER JOIN meta.entite USING(ent_id)  
      WHERE tsm_type = 'liste' AND ent_code = prm_type AND por_id = prm_por_id ORDER BY tom_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

meta\_topmenu\_rename — Renomme une entrée de menu principal.

## Synopsis

```
void meta_topmenu_rename(prm_token, prm_tom_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

```
varchar prm_libelle;
```

## Description

Renomme une entrée de menu principal.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.topmenu SET tom_libelle = prm_libelle WHERE tom_id = prm_tom_id;
END;
```

## Name

meta\_topmenus\_reordonne — Réordonne les entrées du menu principal d'un portail donné.

## Synopsis

```
void meta_topmenus_reordonne(prm_token, prm_por_id);
```

```
int4 prm_token;  
int4 prm_por_id;
```

## Description

Réordonne les entrées du menu principal d'un portail donné.

## Source

```
DECLARE  
  i integer;  
  row RECORD;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  i = 1;  
  FOR row IN  
    SELECT tom_id FROM meta.topmenu WHERE por_id = prm_por_id ORDER BY tom_ordre  
  LOOP  
    UPDATE meta.topmenu SET tom_ordre = i WHERE tom_id = row.tom_id;  
    i = i + 1;  
  END LOOP;  
END;
```



## Name

meta\_topsousmenu\_add\_end — Rajoute une fiche de menu principal.

## Synopsis

```
int4 meta_topsousmenu_add_end(prm_token, prm_tom_id, prm_libelle, prm_icone,  
prm_type, prm_type_id, prm_titre, prm_droit_modif, prm_droit_suppression);
```

```
int4 prm_token;  
int4 prm_tom_id;  
varchar prm_libelle;  
varchar prm_icone;  
varchar prm_type;  
int4 prm_type_id;  
varchar prm_titre;  
bool prm_droit_modif;  
bool prm_droit_suppression;
```

## Description

Rajoute une fiche de menu principal.

## Source

```
DECLARE  
    ret integer;  
    int_ordre integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    PERFORM login._token_assert_interface (prm_token);  
    SELECT MAX(tsm_ordre) + 1 INTO int_ordre FROM meta.topsousmenu WHERE tom_id = prm_tom_id;  
    IF int_ordre ISNULL THEN int_ordre = 0; END IF;  
    INSERT INTO meta.topsousmenu (tom_id, tsm_libelle, tsm_ordre, tsm_icone, tsm_type, tsm_type_id, tsm_titr  
        RETURNING tsm_id INTO ret;  
    RETURN ret;  
END;
```

## Name

meta\_topsousmenu\_delete — Supprime une fiche de menu principal.

## Synopsis

```
void meta_topsousmenu_delete(prm_token, prm_tsm_id);
```

```
int4 prm_token;
```

```
int4 prm_tsm_id;
```

## Description

Supprime une fiche de menu principal.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM procedure.procedure_affectation WHERE tsm_id = prm_tsm_id;
  DELETE FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id;
END;
```

## Name

meta\_topsousmenu\_deplacer\_bas — Déplace une fiche de menu principal vers le bas.

## Synopsis

```
void meta_topsousmenu_deplacer_bas(prm_token, prm_tsm_id);
```

```
int4 prm_token;
```

```
int4 prm_tsm_id;
```

## Description

Déplace une fiche de menu principal vers le bas.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  PERFORM meta.meta_topsousmenus_reordonne (prm_token, (SELECT tom_id FROM meta.topsousmenu WHERE tsm_id
  IF (SELECT tsm_ordre FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id) = (SELECT MAX(tsm_ordre) FROM met
    tom_id = (SELECT tom_id FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id))
  THEN RETURN; END IF;
  UPDATE meta.topsousmenu SET tsm_ordre = tsm_ordre + 1 WHERE tsm_id = prm_tsm_id;
  UPDATE meta.topsousmenu SET tsm_ordre = tsm_ordre - 1 WHERE
    tom_id = (SELECT tom_id FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id) AND
    tsm_ordre = (SELECT tsm_ordre FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id) AND
    tsm_id <> prm_tsm_id;
END;
```

## Name

`meta_topsousmenu_deplacer_haut` — Déplace une fiche de menu principal vers le haut.

## Synopsis

```
void meta_topsousmenu_deplacer_haut(prm_token, prm_tsm_id);
```

```
int4 prm_token;
```

```
int4 prm_tsm_id;
```

## Description

Déplace une fiche de menu principal vers le haut.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  PERFORM meta.meta_topsousmenus_reordonne (prm_token, (SELECT tom_id FROM meta.topsousmenu WHERE tsm_id
  IF (SELECT tsm_ordre FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id) = 1 THEN RETURN; END IF;
  UPDATE meta.topsousmenu SET tsm_ordre = tsm_ordre - 1 WHERE tsm_id = prm_tsm_id;
  UPDATE meta.topsousmenu SET tsm_ordre = tsm_ordre + 1 WHERE
    tom_id = (SELECT tom_id FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id) AND
    tsm_ordre = (SELECT tsm_ordre FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id) AND
    tsm_id <> prm_tsm_id;
END;
```

## Name

meta\_topsousmenu\_events — Retourne la liste des vues événement d'une entrée de menu principal (webdav).

## Synopsis

```
setof topsousmenu meta_topsousmenu_events(prm_token, prm_tom_id);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

## Description

Retourne la liste des vues événement d'une entrée de menu principal (webdav).

## Source

```
DECLARE
  row meta.topsousmenu;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT topsousmenu.* FROM meta.topsousmenu
      WHERE tsm_type = 'event' AND tom_id = prm_tom_id ORDER BY tsm_ordre
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

`meta_topsousmenu_get` — Retourne les informations d'une fiche de menu principal.

## Synopsis

```
topsousmenu meta_topsousmenu_get(prm_token, prm_tsm_id);
```

```
int4 prm_token;
```

```
int4 prm_tsm_id;
```

## Description

Retourne les informations d'une fiche de menu principal.

## Source

```
DECLARE
  ret meta.topsousmenu;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM meta.topsousmenu WHERE tsm_id = prm_tsm_id;
  RETURN ret;
END;
```

## Name

meta\_topsousmenu\_liste — Retourne la liste des fiches d'une entrée de menu principal.

## Synopsis

```
setof topsousmenu meta_topsousmenu_liste(prm_token, prm_tom_id);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

## Description

Retourne la liste des fiches d'une entrée de menu principal.

## Source

```
DECLARE
  row meta.topsousmenu;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT * FROM meta.topsousmenu WHERE tom_id = prm_tom_id ORDER BY tsm_ordre
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

`meta_topsousmenu_liste_type` — Retourne la liste des fiches de type liste pour une catégorie de personne donnée d'une entrée de menu principal d'un type donné (webdav).

## Synopsis

```
setof topsousmenu meta_topsousmenu_liste_type(prm_token, prm_tom_id,  
prm_type);
```

```
int4 prm_token;  
int4 prm_tom_id;  
varchar prm_type;
```

## Description

Retourne la liste des fiches de type liste pour une catégorie de personne donnée d'une entrée de menu principal d'un type donné (webdav).

## Source

```
DECLARE  
  row meta.topsousmenu;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT topsousmenu.* FROM meta.topsousmenu  
    INNER JOIN liste.liste ON liste.lis_id = tsm_type_id  
    INNER JOIN meta.entite USING(ent_id)  
    WHERE tsm_type = 'liste' AND ent_code = prm_type AND tom_id = prm_tom_id ORDER BY tsm_ordre  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```



## Name

meta\_topsousmenu\_rename — Renomme une fiche de menu principal.

## Synopsis

```
void meta_topsousmenu_rename(prm_token, prm_tsm_id, prm_libelle);
```

```
int4 prm_token;
```

```
int4 prm_tsm_id;
```

```
varchar prm_libelle;
```

## Description

Renomme une fiche de menu principal.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  UPDATE meta.topsousmenu SET tsm_libelle = prm_libelle WHERE tsm_id = prm_tsm_id;
END;
```

## Name

meta\_topsousmenu\_set\_droit\_modif — Indique si l'utilisateur a droit de modification sur cette fiche.

## Synopsis

```
void meta_topsousmenu_set_droit_modif(prm_token, prm_id, prm_droit_modif);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

```
bool prm_droit_modif;
```

## Description

Indique si l'utilisateur a droit de modification sur cette fiche.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  UPDATE meta.topsousmenu SET tsm_droit_modif = prm_droit_modif WHERE tsm_id = prm_id;
END;
```

## Name

meta\_topsousmenu\_set\_droit\_suppression — Indique si l'utilisateur a droit de suppression sur cette fiche.

## Synopsis

```
void      meta_topsousmenu_set_droit_suppression(prm_token,      prm_id,  
prm_droit_suppression);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

```
bool prm_droit_suppression;
```

## Description

Indique si l'utilisateur a droit de suppression sur cette fiche.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE meta.topsousmenu SET tsm_droit_suppression = prm_droit_suppression WHERE tsm_id = prm_id;  
END;
```

## Name

meta\_topsousmenu\_update — Modifie les informations d'une fiche de menu principal.

## Synopsis

```
void meta_topsousmenu_update(prm_token, prm_tsm_id, prm_titre, prm_icone,  
prm_type, prm_type_id, prm_sme_id_lien_usager);
```

```
int4 prm_token;  
int4 prm_tsm_id;  
varchar prm_titre;  
varchar prm_icone;  
varchar prm_type;  
int4 prm_type_id;  
int4 prm_sme_id_lien_usager;
```

## Description

Modifie les informations d'une fiche de menu principal.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  UPDATE meta.topsousmenu SET tsm_icone = prm_icone, tsm_type = prm_type, tsm_type_id = prm_type_id, tsm_  
END;
```

## Name

meta\_topsousmenus\_reordonne — Réordonne les fiches d'une entrée de menu principal.

## Synopsis

```
void meta_topsousmenus_reordonne(prm_token, prm_tom_id);
```

```
int4 prm_token;
```

```
int4 prm_tom_id;
```

## Description

Réordonne les fiches d'une entrée de menu principal.

## Source

```
DECLARE
  i integer;
  row RECORD;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  i = 1;
  FOR row IN
    SELECT tsm_id FROM meta.topsousmenu WHERE tom_id = prm_tom_id ORDER BY tsm_ordre
  LOOP
    UPDATE meta.topsousmenu SET tsm_ordre = i WHERE tsm_id = row.tsm_id;
    i = i + 1;
  END LOOP;
END;
```

## Name

metier\_add — Ajoute un métier.

## Synopsis

```
int4 metier_add(prm_token, prm_nom);
```

```
int4 prm_token;  
varchar prm_nom;
```

## Description

Ajoute un métier.

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  INSERT INTO meta.metier (met_nom) VALUES (prm_nom) RETURNING met_id INTO ret;  
  RETURN ret;  
END;
```

## Name

metier\_entite\_liste — Retourne la liste des types de personnes auxquels est affecté un métier.

## Synopsis

```
setof entite metier_entite_liste(prm_token, prm_met_id);
```

```
int4 prm_token;  
int4 prm_met_id;
```

## Description

Retourne la liste des types de personnes auxquels est affecté un métier.

## Source

```
DECLARE  
  row meta.entite;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  FOR row IN  
    SELECT entite.* FROM meta.entite  
    INNER JOIN meta.metier_entite USING(ent_id)  
    WHERE met_id = prm_met_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

metier\_entites\_set — Affecte un métier à des types de personnes.

## Synopsis

```
void metier_entites_set(prm_token, prm_met_id, prm_entites);
```

```
int4 prm_token;
```

```
int4 prm_met_id;
```

```
varchar[] prm_entites;
```

## Description

Affecte un métier à des types de personnes.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM meta.metier_entite WHERE met_id = prm_met_id;
  IF prm_entites NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_entites, 1) LOOP
      INSERT INTO meta.metier_entite (met_id, ent_id) VALUES (prm_met_id, (SELECT ent_id FROM meta.entite
      WHERE ent_id = prm_entites[i]));
    END LOOP;
  END IF;
END;
```



## Name

`metier_get` — Retourne les informations d'un métier.

## Synopsis

```
metier metier_get(prm_token, prm_met_id);
```

```
int4 prm_token;  
int4 prm_met_id;
```

## Description

Retourne les informations d'un métier.

## Source

```
DECLARE  
    ret meta.metier;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM meta.metier WHERE met_id = prm_met_id;  
    RETURN ret;  
END;
```

## Name

metier\_liste — Retourne la liste des métiers affectés à un type de personne.

## Synopsis

```
setof metier metier_liste(prm_token, prm_ent_code);
```

```
int4 prm_token;
```

```
varchar prm_ent_code;
```

## Description

Retourne la liste des métiers affectés à un type de personne.

## Source

```
DECLARE
    row meta.metier;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT metier.* FROM meta.metier
            INNER JOIN meta.metier_entite USING(met_id)
            INNER JOIN meta.entite USING(ent_id)
            WHERE ent_code = prm_ent_code
            ORDER BY metier.met_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

metier\_liste\_details — Retourne la liste détaillée des métiers affectés à un type de personne et assignés à un secteur donné.

## Synopsis

```
setof metier_liste_details metier_liste_details(prm_token, prm_ent_id,  
prm_sec_id);
```

```
int4 prm_token;  
int4 prm_ent_id;  
int4 prm_sec_id;
```

## Description

Retourne la liste détaillée des métiers affectés à un type de personne et assignés à un secteur donné.

## Source

```
DECLARE  
  row meta.metier_liste_details;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT met_id, met_nom, concatenate (DISTINCT sec_nom), concatenate (DISTINCT ent_libelle)  
    FROM meta.metier  
    LEFT JOIN meta.metier_secteur USING(met_id)  
    LEFT JOIN meta.secteur USING(sec_id)  
    LEFT JOIN meta.metier_entite USING(met_id)  
    LEFT JOIN meta.entite USING(ent_id)  
    WHERE (prm_ent_id ISNULL OR ent_id = prm_ent_id)  
    AND (prm_sec_id ISNULL OR sec_id = prm_sec_id)  
    GROUP BY met_id, met_nom  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

metier\_secteur\_liste — Retourne la liste des secteurs assignés à un métier.

## Synopsis

```
setof secteur metier_secteur_liste(prm_token, prm_met_id);
```

```
int4 prm_token;  
int4 prm_met_id;
```

## Description

Retourne la liste des secteurs assignés à un métier.

## Source

```
DECLARE  
  row meta.secteur;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  FOR row IN  
    SELECT secteur.* FROM meta.secteur  
      INNER JOIN meta.metier_secteur USING(sec_id)  
      WHERE met_id = prm_met_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

metier\_secteur\_metier\_liste — Retourne la liste des métiers assignés à un secteur donné.

## Synopsis

```
setof metier metier_secteur_metier_liste(prm_token, prm_sec_id);
```

```
int4 prm_token;
```

```
int4 prm_sec_id;
```

## Description

Retourne la liste des métiers assignés à un secteur donné.

## Source

```
DECLARE
    row meta.metier;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT metier.* FROM meta.metier
            INNER JOIN meta.metier_secteur USING(met_id)
            WHERE sec_id = prm_sec_id
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`metier_secteurs_set` — Assigne des secteurs à un métier.

## Synopsis

```
void metier_secteurs_set(prm_token, prm_met_id, prm_secteurs);
```

```
int4 prm_token;
```

```
int4 prm_met_id;
```

```
varchar[] prm_secteurs;
```

## Description

Assigne des secteurs à un métier.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM meta.metier_secteur WHERE met_id = prm_met_id;
  IF prm_secteurs NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP
      INSERT INTO meta.metier_secteur (met_id, sec_id) VALUES (prm_met_id, (SELECT sec_id FROM meta.secteur WHERE sec_id = prm_secteurs[i]));
    END LOOP;
  END IF;
END;
```

## Name

metier\_supprime — Supprime un métier.

## Synopsis

```
void metier_supprime(prm_token, prm_met_id);
```

```
int4 prm_token;
```

```
int4 prm_met_id;
```

## Description

Supprime un métier.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM meta.metier_secteur WHERE met_id = prm_met_id;
  DELETE FROM meta.metier_entite WHERE met_id = prm_met_id;
  DELETE FROM meta.metier WHERE met_id = prm_met_id;
END;
```

## Name

metier\_update — Modifie les informations d'un métier.

## Synopsis

```
void metier_update(prm_token, prm_met_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_met_id;  
varchar prm_nom;
```

## Description

Modifie les informations d'un métier.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  UPDATE meta.metier SET met_nom = prm_nom WHERE met_id = prm_met_id;  
END;
```

# 11. notes

## 11.1. Description

Données des notes écrites par les utilisateurs. Les utilisateurs peuvent écrire des notes. Ces notes, peuvent, indépendamment :

- être rattachées à un ou plusieurs usagers
- être rattachées à un ou plusieurs groupes d'usagers
- être explicitement à destination d'autres utilisateurs, pour information ou pour action
- être classifiées dans des boîtes thématiques.

L'utilisateur qui a écrit une note peut à tout moment savoir si les utilisateurs destinataires de cette note pour information respectivement action ont marqué cette note comme lue resp. faite.

## 11.2. Tables

### 11.2.1. notes.note

Notes envoyées entre utilisateurs

not\_id (int4)

not\_date\_saisie (timestampz)

not\_date\_evenement (timestampz)

not\_objet (varchar)

not\_texte (text)

uti\_id\_auteur (int4)

Clé étrangère vers utilisateur.uti\_id



eta\_id\_auteur (int4)

Clé étrangère vers etablissement.eta\_id

## Liens vers cette table

- note\_destinataire.not\_id
- note\_groupe.not\_id
- note\_theme.not\_id
- note\_usager.not\_id

### 11.2.2. notes.note\_destinataire

Destinataires d'une note

nde\_id (int4)

not\_id (int4)

Clé étrangère vers note.not\_id

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

nde\_pour\_action (bool)

nde\_pour\_information (bool)

nde\_action\_faite (bool)

nde\_information\_lue (bool)

nde\_supprime (bool)

### 11.2.3. notes.note\_groupe

Rattachement d'une note à un groupe d'usagers

ngr\_id (int4)

not\_id (int4)

Clé étrangère vers note.not\_id

grp\_id (int4)

Clé étrangère vers groupe.grp\_id

### 11.2.4. notes.note\_theme

Classification des notes dans des boîtes thématiques

nth\_id (int4)

not\_id (int4)

Clé étrangère vers note.not\_id

the\_id (int4)

Clé étrangère vers theme.the\_id

### 11.2.5. notes.note\_usager

Rattachement d'une note à un usager

nus\_id (int4)

not\_id (int4)

Clé étrangère vers note.not\_id

per\_id (int4)

Clé étrangère vers personne.per\_id

### 11.2.6. notes.notes

Liste des pages de notes disponibles pour placer dans le menu principal ou usager

nos\_id (int4)

the\_id (int4)

Clé étrangère vers theme.the\_id

nos\_nom (varchar)

### 11.2.7. notes.theme

Liste des boîtes thématiques

the\_id (int4)

the\_nom (varchar)

#### **Liens vers cette table**

- note\_theme.the\_id
- notes.the\_id
- theme\_portail.the\_id

### 11.2.8. notes.theme\_portail

Affectation des boîtes thématiques aux portails

tpo\_id (int4)

the\_id (int4)

Clé étrangère vers theme.the\_id

por\_id (int4)

Clé étrangère vers portail.por\_id

## 11.3. Types

### 11.3.1. notes.note\_destinataire\_derniers\_par\_utilisateur

**11.3.2. notes.note\_destinataires\_liste**

**11.3.3. notes.note\_usagers\_liste**

**11.3.4. notes.notes\_note\_boite\_envoi\_liste**

**11.3.5. notes.notes\_note\_boite\_reception\_liste**

**11.3.6. notes.notes\_note\_mesnotes**

**11.3.7. notes.notes\_theme\_liste\_details**

**11.4. Fonctions**

## Name

`note_destinataire_derniers_par_utilisateur` — Retourne la liste des utilisateurs destinataires de notes de l'utilisateur authentifié, les plus récents en premier.

## Synopsis

```
setof                               note_destinataire_derniers_par_utilisateur
note_destinataire_derniers_par_utilisateur(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des utilisateurs destinataires de notes de l'utilisateur authentifié, les plus récents en premier.  
Entrées :

- `prm_token` : Token d'authentification

Retour :

- `per_id` : Identifiant de la personne
- `libelle` : Nom et prénom de la personne

## Source

```
DECLARE
  uti integer;
  row notes.note_destinataire_derniers_par_utilisateur;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
  FOR row IN
    select DISTINCT per_id, personne_get_libelle (prm_token, per_id), MAX(not_date_saisie) FROM login.uti
    inner join notes.note_destinataire using(uti_id)
    inner join notes.note using(not_id)
    where uti_id_auteur = uti
    group by utilisateur.per_id
    order by MAX(not_date_saisie) desc
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

`note_destinataires_liste` — Retourne la liste des destinataires d'une note.

## Synopsis

```
setof      note_destinataires_liste      note_destinataires_liste(prm_token,
prm_not_id);
```

```
int4 prm_token;
int4 prm_not_id;
```

## Description

Retourne la liste des destinataires d'une note. Entrées :

- `prm_token` : Token d'authentification
- `prm_not_id` : Identifiant de la note

Retour :

- `libelle` : Nom et prénom du destinataire
- `nde_pour_action` : Indique si la note a été envoyée à ce destinataire pour action
- `nde_pour_information` : Indique si la note a été envoyée à ce destinataire pour information
- `nde_action_fait` : Indique si le destinataire a marqué l'action comme faite (si envoyée pour action à ce destinataire)
- `nde_information_lue` : Indique si le destinataire a marqué la note lue (si envoyée pour information à ce destinataire)

## Source

```
DECLARE
    row notes.note_destinataires_liste;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT
            personne_get_libelle (prm_token, per_id),
            nde_pour_action,
            nde_pour_information,
            nde_action_fait,
            nde_information_lue,
            nde_supprime
        FROM personne
        INNER JOIN login.utilisateur USING(per_id)
        INNER JOIN notes.note_destinataire USING(uti_id)
        WHERE not_id = prm_not_id
        ORDER BY personne_get_libelle (prm_token, per_id)
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`note_destinataires_liste_autres` — Retourne la liste des noms et prénoms des destinataires d'une note, autres que l'utilisateur authentifié.

## Synopsis

```
setof varchar note_destinataires_liste_autres(prm_token, prm_not_id);
```

```
int4 prm_token;  
int4 prm_not_id;
```

## Description

Retourne la liste des noms et prénoms des destinataires d'une note, autres que l'utilisateur authentifié. Entrées :

- `prm_token` : Token d'authentification
- `prm_not_id` : Identifiant de la note

Retour :

- Nom et prénom du destinataire

## Source

```
DECLARE  
    uti integer;  
    row RECORD;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;  
    FOR row IN  
        SELECT personne_get_libelle (prm_token, per_id) FROM personne  
        INNER JOIN login.utilisateur USING(per_id)  
        INNER JOIN notes.note_destinataire USING(uti_id)  
        WHERE not_id = prm_not_id AND uti_id <> uti  
        ORDER BY personne_get_libelle (prm_token, per_id)  
    LOOP  
        RETURN NEXT row.personne_get_libelle;  
    END LOOP;  
END;
```

## Name

note\_usagers\_liste — Retourne les usagers rattachés à une note.

## Synopsis

```
setof note_usagers_liste note_usagers_liste(prm_token, prm_not_id);
```

```
int4 prm_token;  
int4 prm_not_id;
```

## Description

Retourne les usagers rattachés à une note. Entrées :

- prm\_token : Token d'authentification
- prm\_not\_id : Identifiant de la note

Retour :

- per\_id : Identifiant de l'utilisateur
- libelle : Nom et prénom de l'utilisateur

## Source

```
DECLARE  
    row notes.note_usagers_liste;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN  
        SELECT per_id, personne_get_libelle (prm_token, per_id) FROM notes.note_usager WHERE not_id = prm_not_id  
        ORDER BY personne_get_libelle (prm_token, per_id)  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

notes\_note\_ajoute — Envoie une note.

## Synopsis

```
int4 notes_note_ajoute(prm_token, prm_date_evenement, prm_objet, prm_texte,
prm_eta_id_auteur, prm_usagers, prm_dests, prm_destsaction, prm_deststheme,
prm_groupes);
```

```
int4 prm_token;
timestampz prm_date_evenement;
varchar prm_objet;
text prm_texte;
int4 prm_eta_id_auteur;
int4[] prm_usagers;
_int4 prm_dests;
_int4 prm_destsaction;
_int4 prm_deststheme;
_int4 prm_groupes;
```

## Description

Envoie une note. Entrées :

- *prm\_token* : Token d'authentification. L'auteur de la note sera l'utilisateur authentifié
- *prm\_date\_evenement* : Date de l'événement décrit par la note
- *prm\_objet* : Objet de la note (facultatid)
- *prm\_texte* : Contenu de la note
- *prm\_eta\_id\_auteur* : Identifiant de l'établissement auquel rattacher la note. Doit être un établissement accessible à l'utilisateur authentifié (TODO : à vérifier)
- *prm\_usagers* : Liste des identifiants des usagers auxquels rattacher la note
- *prm\_dests* : Liste des identifiants des destinataires pour information
- *prm\_destsaction* : Liste des identifiants des destinataires pour action
- *prm\_deststheme* : Liste des identifiants des boîtes thématiques dans lesquelles classer la note
- *prm\_groupes* : Liste des groupes d'usagers auxquels rattacher la note

Retour : L'identifiant de la note.

## Source

```
DECLARE
    uti integer;
    new_not_id integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    INSERT INTO notes.note (not_date_saisie, not_date_evenement, not_objet, not_texte, uti_id_auteur, eta_i
        VALUES (CURRENT_TIMESTAMP, prm_date_evenement, prm_objet, prm_texte, uti, prm_eta_id_auteur)
        RETURNING not_id INTO new_not_id;
    IF prm_usagers NOTNULL THEN
        FOR i IN 1 .. array_upper (prm_usagers, 1) LOOP
            INSERT INTO notes.note_usager (not_id, per_id) VALUES (new_not_id, prm_usagers[i]);
        END LOOP;
```



```
END IF;
IF prm_dests NOTNULL THEN
  FOR i IN 1 .. array_upper (prm_dests, 1) LOOP
    INSERT INTO notes.note_destinataire (not_id, uti_id, nde_pour_information) VALUES (new_not_id, (SEL
  END LOOP;
END IF;
IF prm_destsaction NOTNULL THEN
  FOR i IN 1 .. array_upper (prm_destsaction, 1) LOOP
    INSERT INTO notes.note_destinataire (not_id, uti_id, nde_pour_action) VALUES (new_not_id, (SELECT u
  END LOOP;
END IF;
IF prm_deststHEME NOTNULL THEN
  FOR i IN 1 .. array_upper (prm_deststHEME, 1) LOOP
    INSERT INTO notes.note_THEME (not_id, the_id) VALUES (new_not_id, prm_deststHEME[i]);
  END LOOP;
END IF;
IF prm_groupes NOTNULL THEN
  FOR i IN 1 .. array_upper (prm_groupes, 1) LOOP
    INSERT INTO notes.note_groupe (not_id, grp_id) VALUES (new_not_id, prm_groupes[i]);
  END LOOP;
END IF;
RETURN new_not_id;
END;
```

## Name

notes\_note\_boite\_envoi\_liste — Retourne la liste des notes envoyées par l'utilisateur authentifié.

## Synopsis

```
setof notes_note_boite_envoi_liste notes_note_boite_envoi_liste(prm_token);  
int4 prm_token;
```

## Description

Retourne la liste des notes envoyées par l'utilisateur authentifié. Entrées :

- prm\_token : Token d'authentification

Retour :

- not\_id : Identifiant de la note
- not\_date\_saisie : Date de saisie de la note
- not\_date\_evenement : Date de l'événement décrit par la note
- not\_objet : Objet de la note
- not\_texte : Contenu de la note
- eta\_id\_auteur : Identifiant de l'établissement auquel est rattachée la note

## Source

```
DECLARE  
    uti integer;  
    row notes.notes_note_boite_envoi_liste;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;  
    FOR row IN  
        SELECT not_id, not_date_saisie, not_date_evenement, not_objet, not_texte, eta_id_auteur  
            FROM notes.note  
            WHERE uti_id_auteur = uti  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

notes\_note\_boite\_reception\_liste — Retourne la liste des notes dont l'utilisateur authentifié est destinataire.

## Synopsis

```
setof          notes_note_boite_reception_liste
notes_note_boite_reception_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des notes dont l'utilisateur authentifié est destinataire. Entrées :

- `prm_token` : Token d'authentification

Retour :

- `not_id` : Identifiant de la note
- `not_date_saisie` : Date de saisie de la note
- `not_date_evenement` : Date de l'événement décrit par la note
- `not_objet` : Objet de la note
- `not_texte` : Contenu de la note
- `uti_id_auteur` : Identifiant de l'auteur de la note
- `eta_id_auteur` : Identifiant de l'établissement auquel est rattachée la note
- `nde_id` : Identifiant des informations liées au destinataire de cette note
- `nde_pour_action` : Note pour action pour le destinataire authentifié
- `nde_pour_information` : Note pour information pour le destinataire authentifié
- `nde_action_fait` : L'action a été marquée comme faite par le destinataire authentifié
- `nde_information_lue` : La note a été marquée comme lue par le destinataire authentifié

## Source

```
DECLARE
    uti integer;
    row notes.notes_note_boite_reception_liste;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    FOR row IN
        SELECT not_id, not_date_saisie, not_date_evenement, not_objet, not_texte, uti_id_auteur, eta_id_auteur,
            nde_id, nde_pour_action, nde_pour_information, nde_action_fait, nde_information_lue
        FROM notes.note
        INNER JOIN notes.note_destinataire USING(not_id)
        WHERE note_destinataire.uti_id = uti AND NOT nde_supprime
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`notes_note_boite_reception_nombre_non_lu` — Retourne le nombre de notes dont l'utilisateur authentifié est destinataire et qui n'ont pas encore été marquées comme lues ou faite.

## Synopsis

```
int4 notes_note_boite_reception_nombre_non_lu(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne le nombre de notes dont l'utilisateur authentifié est destinataire et qui n'ont pas encore été marquées comme lues ou faite. Entrées :

- `prm_token` : Token d'authentification

Retour :

- le nombre de notes.

## Source

```
DECLARE
  uti integer;
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
  SELECT count(*) INTO ret
  FROM notes.note
  INNER JOIN notes.note_destinataire USING(not_id)
  WHERE note_destinataire.uti_id = uti AND NOT nde_supprime AND NOT nde_information_lue AND NOT nde_act
  RETURN ret;
END;
```

## Name

notes\_note\_destinataire\_ajoute\_forward\_pour\_info — Ajoute un destinataire pour information à une note (forward de la note).

## Synopsis

```
void          notes_note_destinataire_ajoute_forward_pour_info(prm_token,  
prm_not_id, prm_per_id);
```

```
int4 prm_token;  
int4 prm_not_id;  
int4 prm_per_id;
```

## Description

Ajoute un destinataire pour information à une note (forward de la note). Entrées :

- *prm\_token* : Token d'authentification
- *prm\_not\_id* : Identifiant de la note
- *prm\_per\_id* : Identifiant de la personne destinataire pour information

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  INSERT INTO notes.note_destinataire (not_id, uti_id, nde_pour_information)  
    VALUES (prm_not_id, (SELECT uti_id FROM login.utilisateur WHERE per_id = prm_per_id), TRUE);  
END;
```

## Name

notes\_note\_destinataire\_marque\_done — Marque un message comme lu/fait.

## Synopsis

```
void notes_note_destinataire_marque_done(prm_token, prm_nde_id);
```

```
int4 prm_token;
```

```
int4 prm_nde_id;
```

## Description

Marque un message comme lu/fait. TODO : Vérifier que l'utilisateur authentifié est bien le destinataire de `prm_nde_id` Entrées :

- `prm_token` : Token d'authentification
- `prm_nde_id` : Identifiant des informations liées au destinataire de cette note (obtenu avec la fonction `notes_note_boite_reception_liste`)

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  UPDATE notes.note_destinataire SET nde_information_lue = TRUE WHERE nde_pour_information AND nde_id = prm_nde_id;
  UPDATE notes.note_destinataire SET nde_action_faite = TRUE WHERE nde_pour_action AND nde_id = prm_nde_id;
END;
```

## Name

notes\_note\_mesnotes — Retourne la liste des notes intéressant l'utilisateur authentifié. Les notes "intéressantes" sont celles rattachées aux usagers dont l'utilisateur a accès (via les groupes d'usagers/groupes d'utilisateurs).

## Synopsis

```
setof notes_note_mesnotes notes_note_mesnotes(prm_token, prm_grp_id,
prm_nos_id);
```

```
int4 prm_token;
int4 prm_grp_id;
int4 prm_nos_id;
```

## Description

Retourne la liste des notes intéressant l'utilisateur authentifié. Les notes "intéressantes" sont celles rattachées aux usagers dont l'utilisateur a accès (via les groupes d'usagers/groupes d'utilisateurs). Entrées :

- *prm\_token* : Token d'authentification
- *prm\_grp\_id* : Filtre sur un groupe d'usagers en particulier, ou NULL
- *prm\_nos\_id* : Identifiant de pages de notes (pour appliquer des filtres définis dans cette page)

## Source

```
DECLARE
    uti integer;
    row notes.notes_note_mesnotes;
    the integer DEFAULT NULL;
BEGIN
    PERFORM login.token_assert (prm_token, FALSE, FALSE);
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;
    IF prm_nos_id NOTNULL THEN
        SELECT the_id INTO the FROM notes.notes WHERE nos_id = prm_nos_id;
    END IF;
    IF the ISNULL THEN
        FOR row IN
            select DISTINCT note.* FROM notes.note
                inner join notes.note_usager using(not_id)
                inner join login.utilisateur_usagers_liste (prm_token, prm_grp_id, false) on utilisateur_usagers_
        LOOP
            RETURN NEXT row;
        END LOOP;
    ELSE
        FOR row IN
            select DISTINCT note.* FROM notes.note
                inner join notes.note_usager using(not_id)
                inner join notes.note_theme USING(not_id)
                inner join login.utilisateur_usagers_liste (prm_token, prm_grp_id, false) on utilisateur_usagers_
                WHERE the_id = the
        LOOP
            RETURN NEXT row;
        END LOOP;
    END IF;
END;
```

## Name

notes\_note\_supprime — Supprime une note.

## Synopsis

```
void notes_note_supprime(prm_token, prm_not_id);
```

```
int4 prm_token;  
int4 prm_not_id;
```

## Description

Supprime une note. Entrées :

- *prm\_token* : Token d'authentification
- *not\_id* : Identifiant de la note à supprimer

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  DELETE FROM notes.note_destinataire WHERE not_id = prm_not_id;  
  DELETE FROM notes.note_groupe WHERE not_id = prm_not_id;  
  DELETE FROM notes.note_usager WHERE not_id = prm_not_id;  
  DELETE FROM notes.note_theme WHERE not_id = prm_not_id;  
  DELETE FROM notes.note WHERE not_id = prm_not_id;  
END;
```



## Name

notes\_note\_usager\_liste — Retourne les notes rattachées à un usager.

## Synopsis

```
setof note notes_note_usager_liste(prm_token, prm_per_id, prm_nos_id);
```

```
int4 prm_token;  
int4 prm_per_id;  
int4 prm_nos_id;
```

## Description

Retourne les notes rattachées à un usager. Entrées :

- prm\_token : Token d'authentification
- prm\_per\_id : Identifiant de l'usager

TODO : Vérifier que l'utilisateur authentifié a droit d'accès à cet usager

## Source

```
DECLARE  
    the integer;  
    row notes.note;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT the_id INTO the FROM notes.notes WHERE nos_id = prm_nos_id;  
    IF the ISNULL THEN  
        FOR row IN  
            SELECT note.* FROM notes.note  
                INNER JOIN notes.note_usager USING(not_id)  
                WHERE per_id = prm_per_id  
        LOOP  
            RETURN NEXT row;  
        END LOOP;  
    ELSE  
        FOR row IN  
            SELECT note.* FROM notes.note  
                INNER JOIN notes.note_usager USING(not_id)  
                INNER JOIN notes.note_theme USING(not_id)  
                WHERE per_id = prm_per_id AND the_id = the  
        LOOP  
            RETURN NEXT row;  
        END LOOP;  
    END IF;  
END;
```

## Name

notes\_notes\_get — Retourne les informations sur la configuration d'une page de notes.

## Synopsis

```
notes notes_notes_get(prm_token, prm_nos_id);
```

```
int4 prm_token;  
int4 prm_nos_id;
```

## Description

Retourne les informations sur la configuration d'une page de notes. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_nos\_id* : Identifiant de la configuration de la page de notes.

## Source

```
DECLARE  
    ret notes.notes;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM notes.notes WHERE nos_id = prm_nos_id;  
    RETURN ret;  
END;
```

## Name

notes\_notes\_liste — Retourne la liste des informations de configuration de pages de notes, à placer dans le menu principal ou usager.

## Synopsis

```
setof notes notes_notes_liste(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste des informations de configuration de pages de notes, à placer dans le menu principal ou usager.  
Remarque : Nécessite le droit de configuration de l'interface

## Source

```
DECLARE
    row notes.notes;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    PERFORM login._token_assert_interface (prm_token);
    FOR row IN SELECT * FROM notes.notes ORDER BY nos_nom LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

notes\_notes\_supprime — Supprime une configuration de page de notes.

## Synopsis

```
void notes_notes_supprime(prm_token, prm_nos_id);
```

```
int4 prm_token;
```

```
int4 prm_nos_id;
```

## Description

Supprime une configuration de page de notes. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_nos\_id* : Identifiant de la configuration de page de notes

Remarque : Nécessite le droit de configuration de l'interface

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  PERFORM login._token_assert_interface (prm_token);
  DELETE FROM notes.notes WHERE nos_id = prm_nos_id;
END;
```

## Name

notes\_notes\_update — Modifie les informations de configuration d'une page de notes ou crée une nouvelle configuration.

## Synopsis

```
int4 notes_notes_update(prm_token, prm_nos_id, prm_nom, prm_the_id);
```

```
int4 prm_token;  
int4 prm_nos_id;  
varchar prm_nom;  
int4 prm_the_id;
```

## Description

Modifie les informations de configuration d'une page de notes ou crée une nouvelle configuration. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_nos\_id* : Identifiant de la configuration de page à modifier ou NULL pour créer une nouvelle configuration
- *prm\_nom* : Nouveau nom de page
- *prm\_the\_id* : Identifiant de la boîte thématique permettant de filtrer les notes sur cette page

Remarque : Nécessite le droit de configuration de l'interface

## Source

```
DECLARE  
  ret integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  PERFORM login._token_assert_interface (prm_token);  
  IF prm_nos_id NOTNULL THEN  
    UPDATE notes.notes SET nos_nom = prm_nom, the_id = prm_the_id WHERE nos_id = prm_nos_id;  
    RETURN prm_nos_id;  
  ELSE  
    INSERT INTO notes.notes (the_id, nos_nom) VALUES (prm_the_id, prm_nom) RETURNING nos_id INTO ret;  
    RETURN ret;  
  END IF;  
END;
```

## Name

notes\_theme\_get — Retourne les informations sur une boîte thématique.

## Synopsis

```
theme notes_theme_get(prm_token, prm_the_id);
```

```
int4 prm_token;  
int4 prm_the_id;
```

## Description

Retourne les informations sur une boîte thématique. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_the\_id* : Identifiant de la boîte thématique

## Source

```
DECLARE  
    ret notes.theme;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM notes.theme WHERE the_id = prm_the_id;  
    RETURN ret;  
END;
```

## Name

notes\_theme\_liste\_details — Retourne le détail des informations des boîtes thématiques affectées à un portail.

## Synopsis

```
setof notes_theme_liste_details notes_theme_liste_details(prm_token,  
prm_por_id);
```

```
int4 prm_token;  
int4 prm_por_id;
```

## Description

Retourne le détail des informations des boîtes thématiques affectées à un portail. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_por\_id* : Identifiant du portail

Retour :

- *the\_id* : Identifiant de la boîte thématique
- *the\_nom* : Nom de la boîte thématique
- *portails* : Liste des noms de portails auxquels sont affectés cette boîte thématique

## Source

```
DECLARE  
  row notes_theme_liste_details ;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT DISTINCT the_id, the_nom, concatenate (por_libelle) FROM notes.theme  
      INNER JOIN notes.theme_portail USING(the_id)  
      INNER JOIN meta.portail USING(por_id)  
    WHERE (prm_por_id ISNULL OR prm_por_id = por_id)  
    GROUP BY the_id, the_nom  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

notes\_theme\_portail\_liste — Retourne la liste des portails auxquels est affectée une boîte thématique.

## Synopsis

```
setof portail notes_theme_portail_liste(prm_token, prm_the_id);
```

```
int4 prm_token;
```

```
int4 prm_the_id;
```

## Description

Retourne la liste des portails auxquels est affectée une boîte thématique. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_the\_id* : Identifiant de la boîte thématique

## Source

```
DECLARE
    row meta.portail;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT portail.* FROM meta.portail INNER JOIN notes.theme_portail USING(por_id) WHERE the_id = prm_th
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```



## Name

notes\_theme\_supprime — Supprime une boîte thématique.

## Synopsis

```
void notes_theme_supprime(prm_token, prm_the_id);
```

```
int4 prm_token;
```

```
int4 prm_the_id;
```

## Description

Supprime une boîte thématique. Entrées :

- *prm\_token* : Token d'authentification
- *prm\_the\_id* : Identifiant de la boîte thématique

Remarque : Nécessite le droit de configuration "Réseau"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, TRUE);
  DELETE FROM notes.theme_portail WHERE the_id = prm_the_id;
  DELETE FROM notes.theme WHERE the_id = prm_the_id;
END;
```

## Name

notes\_theme\_update — Modifie les informations d'une boîte thématique.

## Synopsis

```
int4 notes_theme_update(prm_token, prm_the_id, prm_the_nom, prm_portails);
```

```
int4 prm_token;
int4 prm_the_id;
varchar prm_the_nom;
int4[] prm_portails;
```

## Description

Modifie les informations d'une boîte thématique. Entrées :

- prm\_token : Token d'authentification
- prm\_the\_id : Identifiant de la boîte thématique à modifier
- prm\_the\_nom : Nom de la boîte thématique
- prm\_portails : Tableau d'identifiants de portails auxquels affecter la boîte thématique

Remarque : Nécessite le droit de configuration "Réseau"

## Source

```

DECLARE
    ret integer;
BEGIN
    PERFORM login_token_assert (prm_token, FALSE, TRUE);
    IF prm_the_id ISNULL THEN
        INSERT INTO notes.theme (the_nom) VALUES (prm_the_nom) RETURNING the_id INTO ret;
    ELSE
        UPDATE notes.theme SET the_nom = prm_the_nom WHERE the_id = prm_the_id;
        ret = prm_the_id;
    END IF;
    DELETE FROM notes.theme_portail WHERE the_id = ret;
    IF prm_portails NOTNULL THEN
        FOR i IN 1 .. array_upper(prm_portails, 1) LOOP
            INSERT INTO notes.theme_portail(the_id, por_id) VALUES (ret, prm_portails[i]);
        END LOOP;
    END IF;
    RETURN ret;
END;
```

## 12. permission

### 12.1. Description

Enregistrement des diverses permissions.

### 12.2. Tables

#### 12.2.1. permission.droit\_ajout\_entite\_portail

Droit pour les utilisateurs d'ajouter des personnes d'une certaine catégorie (usager, personnel, contact, famille) par portail.

daj\_id (int4)

ent\_code (varchar)

Clé étrangère vers entite.ent\_code

por\_id (int4)

Clé étrangère vers portail.por\_id

daj\_droit (bool)

## 12.3. Fonctions

## Name

`droit_ajout_entite_portail_liste_par_portail` — Retourne la liste des droits d'ajout de personne pour un portail donné.

## Synopsis

```
setof                                droit_ajout_entite_portail
droit_ajout_entite_portail_liste_par_portail(prm_token, prm_por_id);
```

```
int4 prm_token;
int4 prm_por_id;
```

## Description

Retourne la liste des droits d'ajout de personne pour un portail donné.

## Source

```
DECLARE
    row permission.droit_ajout_entite_portail;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN SELECT * FROM permission.droit_ajout_entite_portail WHERE por_id = prm_por_id LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`droit_ajout_entite_portail_set` — Indique le droit d'ajouter une personne d'une certaine catégorie depuis un portail donné.

## Synopsis

```
void droit_ajout_entite_portail_set(prm_token, prm_ent_code, prm_por_id,  
prm_droit);
```

```
int4 prm_token;  
varchar prm_ent_code;  
int4 prm_por_id;  
bool prm_droit;
```

## Description

Indique le droit d'ajouter une personne d'une certaine catégorie depuis un portail donné.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, TRUE);  
  UPDATE permission.droit_ajout_entite_portail SET daj_droit = prm_droit WHERE ent_code = prm_ent_code AND  
  IF NOT FOUND THEN  
    INSERT INTO permission.droit_ajout_entite_portail (ent_code, por_id, daj_droit) VALUES (prm_ent_code,  
  END IF;  
END;
```

# 13. procedure

## 13.1. Description

Documentations à afficher sur diverses pages.

## 13.2. Tables

### 13.2.1. procedure.procedure

Contenu de la documentation.

`pro_id` (int4)

`pro_titre` (varchar)

`pro_contenu` (text)

### Liens vers cette table

- `procedure_affectation.pro_id`

### 13.2.2. procedure.procedure\_affectation

Affichage de la documentation sur une page donnée.

`paf_id` (int4)

`pro_id` (int4)

Clé étrangère vers `procedure.pro_id`

tsm\_id (int4)

Clé étrangère vers topsousmenu.tsm\_id

sme\_id (int4)

Clé étrangère vers sousmenu.sme\_id

## **13.3. Types**

### **13.3.1. procedure.procedure\_procedure\_details**

## **13.4. Fonctions**

## Name

`procedure_liste` — Retourne la liste des procédures affectées à une page donnée.

## Synopsis

```
setof procedure procedure_liste(prm_token, prm_id, prm_table);
```

```
int4 prm_token;
```

```
int4 prm_id;
```

```
varchar prm_table;
```

## Description

Retourne la liste des procédures affectées à une page donnée.

## Source

```
DECLARE
    row procedure.procedure;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    CASE prm_table
        WHEN 'tsm' THEN
            FOR row IN
                SELECT procedure.pro_id, procedure.pro_titre
                   FROM procedure.procedure
                   INNER JOIN procedure.procedure_affectation USING(pro_id)
                   WHERE tsm_id = prm_id
            LOOP
                RETURN NEXT row;
            END LOOP;
        WHEN 'sme' THEN
            FOR row IN
                SELECT procedure.pro_id, procedure.pro_titre
                   FROM procedure.procedure
                   INNER JOIN procedure.procedure_affectation USING(pro_id)
                   WHERE sme_id = prm_id
            LOOP
                RETURN NEXT row;
            END LOOP;
    END CASE;
END;
```

## Name

procedure\_procedure\_affectation\_ajoute — Affecte une procédure à une page.

## Synopsis

```
int4      procedure_procedure_affectation_ajoute(prm_token,      prm_pro_id,  
prm_tsm_id, prm_sme_id);
```

```
int4 prm_token;  
int4 prm_pro_id;  
int4 prm_tsm_id;  
int4 prm_sme_id;
```

## Description

Affecte une procédure à une page.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, TRUE, FALSE);  
    INSERT INTO procedure.procedure_affectation (pro_id, tsm_id, sme_id) VALUES (prm_pro_id, prm_tsm_id, prm_sme_id)  
        RETURNING paf_id INTO ret;  
    RETURN ret;  
END;
```



## Name

`procedure_procedure_affectation_detail` — Retourne les informations sur l'affectation d'une procédure à une page.

## Synopsis

```
vvarchar procedure_procedure_affectation_detail(prm_token, prm_paf_id);
```

```
int4 prm_token;
```

```
int4 prm_paf_id;
```

## Description

Retourne les informations sur l'affectation d'une procédure à une page.

## Source

```
DECLARE
    ret varchar;
    paf procedure.procedure_affectation;
BEGIN
    PERFORM login._token_assert (prm_token, TRUE, FALSE);
    SELECT * INTO paf FROM procedure.procedure_affectation WHERE paf_id = prm_paf_id;
    IF paf.tsm_id NOTNULL THEN
        ret = (SELECT por_libelle FROM meta.portail INNER JOIN meta.topmenu USING(por_id) INNER JOIN meta.to
            || ' > Menu principal > '
            || (SELECT tom_libelle FROM meta.topmenu WHERE tom_id = (SELECT tom_id FROM meta.topsousmenu WHERE
            || ' > '
            || (SELECT tsm_libelle FROM meta.topsousmenu WHERE tsm_id = paf.tsm_id);
        RETURN ret;
    END IF;
    IF paf.sme_id NOTNULL THEN
        ret = (SELECT por_libelle FROM meta.portail INNER JOIN meta.menu USING(por_id) INNER JOIN meta.sousme
            || ' > Menu '
            || (SELECT ent_libelle FROM meta.entite INNER JOIN meta.menu USING(ent_id) INNER JOIN meta.sousmenu
            || ' > '
            || (SELECT men_libelle FROM meta.menu WHERE men_id = (SELECT men_id FROM meta.sousmenu WHERE sme_id
            || ' > '
            || (SELECT sme_libelle FROM meta.sousmenu WHERE sme_id = paf.sme_id);
        RETURN ret;
    END IF;
END;
```

## Name

`procedure_procedure_affectation_liste` — Retourne la liste des affectations d'une procédure.

## Synopsis

```
setof procedure_affectation  
procedure_procedure_affectation_liste(prm_token, prm_pro_id);
```

```
int4 prm_token;  
int4 prm_pro_id;
```

## Description

Retourne la liste des affectations d'une procédure.

## Source

```
DECLARE  
  row procedure.procedure_affectation;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  FOR row IN  
    SELECT * FROM procedure.procedure_affectation WHERE pro_id = prm_pro_id ORDER BY paf_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`procedure_procedure_affectation_supprime` — Supprime une affectation d'une procédure à une page.

## Synopsis

```
void procedure_procedure_affectation_supprime(prm_token, prm_paf_id);
```

```
int4 prm_token;
```

```
int4 prm_paf_id;
```

## Description

Supprime une affectation d'une procédure à une page.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM procedure.procedure_affectation WHERE paf_id = prm_paf_id;
END;
```

## Name

procedure\_procedure\_details — Retourne la liste détaillée des procédures.

## Synopsis

```
setof procedure_procedure_details procedure_procedure_details(prm_token);
```

```
int4 prm_token;
```

## Description

Retourne la liste détaillée des procédures.

## Source

```
DECLARE
    row procedure.procedure_procedure_details;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT pro_id, pro_titre,
            (SELECT COUNT(*) FROM procedure.procedure_affectation WHERE procedure_affectation.pro_id = procedur
            FROM procedure.procedure
            ORDER BY pro_titre
        LOOP
            RETURN NEXT row;
        END LOOP;
END;
```

## Name

`procedure_procedure_get` — Retourne les informations sur une procédure.

## Synopsis

```
procedure procedure_procedure_get(prm_token, prm_pro_id);
```

```
int4 prm_token;  
int4 prm_pro_id;
```

## Description

Retourne les informations sur une procédure.

## Source

```
DECLARE  
  ret procedure.procedure;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  SELECT * INTO ret FROM procedure.procedure WHERE pro_id = prm_pro_id;  
  RETURN ret;  
END;
```

## Name

procedure\_procedure\_supprime — Supprime une procédure.

## Synopsis

```
void procedure_procedure_supprime(prm_token, prm_pro_id);
```

```
int4 prm_token;
```

```
int4 prm_pro_id;
```

## Description

Supprime une procédure.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM procedure.procedure_affectation WHERE pro_id = prm_pro_id;
  DELETE FROM procedure.procedure WHERE pro_id = prm_pro_id;
END;
```

## Name

procedure\_procedure\_update — Modifie les informations d'une procédure.

## Synopsis

```
int4 procedure_procedure_update(prm_token, prm_pro_id, prm_pro_titre,
prm_pro_contenu);
```

```
int4 prm_token;
int4 prm_pro_id;
varchar prm_pro_titre;
text prm_pro_contenu;
```

## Description

Modifie les informations d'une procédure.

## Source

```

DECLARE
    ret integer;
BEGIN
    PERFORM login.token_assert (prm_token, TRUE, FALSE);
    IF prm_pro_id NOTNULL THEN
        UPDATE procedure.procedure SET pro_titre = prm_pro_titre, pro_contenu = prm_pro_contenu WHERE pro_id
        RETURN prm_pro_id;
    ELSE
        INSERT INTO procedure.procedure (pro_titre, pro_contenu) VALUES (prm_pro_titre, prm_pro_contenu) RETU
        RETURN ret;
    END IF;
END;
```

# 14. public

## 14.1. Description

Données des établissements, groupes et personnes.

## 14.2. Tables

### 14.2.1. public.adresse

Adresse et moyens de contact d'un établissement/partenaire.

adr\_id (int4)  
Identifiant de l'adresse

adr\_adresse (varchar)  
Ligne d'adresse

adr\_cp (varchar)  
Code postal

adr\_ville (varchar)  
Ville

adr\_tel (varchar)  
Téléphone

adr\_fax (varchar)  
Fax

adr\_email (varchar)  
Adresse email de contact de l'établissement

adr\_web (varchar)  
Site web de l'établissement

### **Liens vers cette table**

- etablissement.adr\_id

#### **14.2.2. public.etablissement**

Etablissement (internes) et partenaires (externes).cat\_id est NULL pour les partenaires, non NULL pour les établissements.

eta\_id (int4)  
Identifiant de l'établissement/partenaire

cat\_id (int4)  
Clé étrangère vers categorie.cat\_id  
  
Identifiant de la catégorie de l'établissement, NULL pour un partenaire

eta\_nom (varchar)  
Nom de l'établissement/partenaire

adr\_id (int4)  
Clé étrangère vers adresse.adr\_id  
  
Identifiant de l'adresse

### **Liens vers cette table**

- document\_type\_etablissement.eta\_id
- event\_type\_etablissement.eta\_id
- note.eta\_id\_auteur
- etablissement\_secteur.eta\_id
- etablissement\_secteur\_edit.eta\_id
- groupe.eta\_id
- personne\_etablissement.eta\_id

#### **14.2.3. public.etablissement\_secteur**

Liste des secteurs couverts par les établissements/partenaires.

ets\_id (int4)  
Identifiant de la relation

eta\_id (int4)  
Clé étrangère vers etablissement.eta\_id  
  
Identifiant de l'établissement/partenaire

sec\_id (int4)  
Clé étrangère vers secteur.sec\_id



Identifiant du secteur

#### 14.2.4. public.etablissement\_secteur\_edit

Liste des secteurs pour lesquels les utilisateurs ont le droit de rajouter des groupes à un établissement/partenaire.

ese\_id (int4)

Identifiant de la relation

eta\_id (int4)

Clé étrangère vers etablissement.eta\_id

Identifiant de l'établissement

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

Identifiant du secteur

#### 14.2.5. public.groupe

Groupe d'usagers

grp\_id (int4)

Identifiant du groupe

grp\_nom (varchar)

Nom du groupe

grp\_complet (bool)

Indique s'il est possible d'affecter des usagers à ce groupe (non utilisé)

eta\_id (int4)

Clé étrangère vers etablissement.eta\_id

Identifiant de l'établissement/partenaire auquel appartient ce groupe

grp\_debut (date)

Date de début d'activité du groupe

grp\_fin (date)

Date de fin d'activité du groupe

grp\_hebergement\_adresse (varchar)

grp\_hebergement\_cp (varchar)

grp\_hebergement\_ville (varchar)

grp\_pedagogie\_type (int4)

grp\_pedagogie\_niveau (int4)

grp\_pedagogie\_contact (int4)

grp\_notes (varchar)

Notes sur le groupe

grp\_sante\_contact (int4)

grp\_emploi\_contact (int4)

grp\_culture\_contact (int4)  
grp\_education\_contact (int4)  
grp\_hebergement\_contact (int4)  
grp\_justice\_contact (int4)  
grp\_social\_contact (int4)  
grp\_sport\_contact (int4)  
grp\_transport\_contact (int4)  
grp\_decideur\_contact (int4)  
grp\_financeur\_contact (int4)  
grp\_prise\_en\_charge\_contact (int4)  
grp\_divertissement\_contact (int4)  
grp\_protection\_juridique\_contact (int4)  
grp\_restauracion\_contact (int4)  
grp\_entretien\_contact (int4)  
grp\_aide\_a\_la\_personne\_contact (int4)  
grp\_social\_type (int4)  
grp\_education\_type (int4)  
grp\_sante\_type (int4)  
grp\_justice\_type (int4)  
grp\_emploi\_type (int4)  
grp\_sport\_type (int4)  
grp\_culture\_type (int4)  
grp\_transport\_type (int4)  
grp\_decideur\_type (int4)  
grp\_hebergement\_type (int4)  
grp\_financeur\_type (int4)  
grp\_prise\_en\_charge\_type (int4)  
grp\_divertissement\_type (int4)  
grp\_protection\_juridique\_type (int4)  
grp\_restauracion\_type (int4)  
grp\_entretien\_type (int4)  
grp\_aide\_a\_la\_personne\_type (int4)

grp\_aide\_financiere\_directe\_contact (int4)  
grp\_aide\_financiere\_directe\_type (int4)  
grp\_equipement\_personnel\_contact (int4)  
grp\_equipement\_personnel\_type (int4)  
grp\_famille\_contact (int4)  
grp\_famille\_type (int4)  
grp\_projet\_contact (int4)  
grp\_projet\_type (int4)  
grp\_sejour\_contact (int4)  
grp\_sejour\_type (int4)  
grp\_soins\_infirmiers\_contact (int4)  
grp\_soins\_infirmiers\_type (int4)  
grp\_dietetique\_contact (int4)  
grp\_dietetique\_type (int4)  
grp\_ergotherapie\_contact (int4)  
grp\_ergotherapie\_type (int4)  
grp\_physiotherapie\_contact (int4)  
grp\_physiotherapie\_type (int4)  
grp\_kinesitherapie\_contact (int4)  
grp\_kinesitherapie\_type (int4)  
grp\_orthophonie\_contact (int4)  
grp\_orthophonie\_type (int4)  
grp\_psychomotricite\_contact (int4)  
grp\_psychomotricite\_type (int4)  
grp\_psychologie\_contact (int4)  
grp\_psychologie\_type (int4)  
grp\_droits\_de\_sejour\_contact (int4)  
grp\_droits\_de\_sejour\_type (int4)  
grp\_aide\_formalites\_contact (int4)  
grp\_aide\_formalites\_type (int4)

### **Liens vers cette table**

- grouputil\_groupe.grp\_id

- note\_groupe.grp\_id
- groupe\_info\_secteur.grp\_id
- groupe\_secteur.grp\_id
- personne\_groupe.grp\_id

#### **14.2.6. public.groupe\_info\_secteur**

Indique sur quel champ de fiche usager est faite l'affectation de l'utilisateur à un groupe pour un secteur donné.

gis\_id (int4)

Identifiant de la relation

grp\_id (int4)

Clé étrangère vers groupe.grp\_id

Identifiant du groupe

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

Identifiant du secteur

inf\_id (int4)

Clé étrangère vers info.inf\_id

Identifiant du champ

#### **14.2.7. public.groupe\_secteur**

Liste des secteurs couverts par un groupe d'utilisateurs.

grs\_id (int4)

Identifiant de la relation

grp\_id (int4)

Clé étrangère vers groupe.grp\_id

Identifiant du groupe

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

Identifiant du secteur

#### **14.2.8. public.personne**

Information de base sur les usagers/personnels/contacts/membres de famille.

per\_id (int4)

Identifiant de la personne

ent\_code (varchar)

Clé étrangère vers entite.ent\_code

Code du type de personne

#### **Liens vers cette table**

- document.per\_id\_responsable

- document\_usager.per\_id\_usager
- event\_personne.per\_id
- fiche.per\_id
- utilisateur.per\_id
- note\_usager.per\_id
- personne\_etablissement.per\_id
- personne\_groupe.per\_id
- personne\_info.per\_id
- personne\_info\_lien\_familial.per\_id\_parent

### 14.2.9. public.personne\_etablissement

Appartenance d'un usager à un établissement. Cette table est mise à jour par la fonction `personne_etablissement_update()`

`per_id` (int4)

Clé étrangère vers `personne.per_id`

Identifiant de l'usager

`eta_id` (int4)

Clé étrangère vers `etablissement.eta_id`

Identifiant de l'établissement

### 14.2.10. public.personne\_groupe

Affectation d'un usager à un groupe d'usagers.

`peg_id` (int4)

Identifiant de la relation

`per_id` (int4)

Clé étrangère vers `personne.per_id`

Identifiant de l'usager

`grp_id` (int4)

Clé étrangère vers `groupe.grp_id`

Identifiant du groupe

`peg_debut` (date)

Date de début d'affectation de l'usager au groupe

`peg_fin` (date)

Date de fin d'affectation

`peg_cycle_statut` (int4)

Statut du cycle : 1: Demandé 2: Accepté 3: Terminé-1: Refusé

`peg_cycle_date_demande` (date)

Date de demande d'affectation

peg\_cycle\_date\_demande\_renouvellement (date)  
Date de demande de renouvellement de l'affectation

peg\_hebergement\_chambre (varchar)

peg\_notes (text)  
Notes sur l'affectation

\_inf\_id (int4)  
Clé étrangère vers info.inf\_id  
  
(non utilisé)

peg\_decideur\_financeur (int4)

### 14.2.11. public.personne\_info

Valeur d'un champ pour une personne.

pin\_id (int4)  
Identifiant de la relation

per\_id (int4)  
Clé étrangère vers personne.per\_id  
  
Identifiant de la personne

inf\_code (varchar)  
Clé étrangère vers info.inf\_code  
  
Code du champ

#### Liens vers cette table

- [personne\\_info\\_boolean.pin\\_id](#)
- [personne\\_info\\_date.pin\\_id](#)
- [personne\\_info\\_integer.pin\\_id](#)
- [personne\\_info\\_integer2.pin\\_id](#)
- [personne\\_info\\_lien\\_familial.pin\\_id](#)
- [personne\\_info\\_text.pin\\_id](#)
- [personne\\_info\\_varchar.pin\\_id](#)

### 14.2.12. public.personne\_info\_boolean

Valeur d'un champ de type "case à cocher" pour une personne.

pib\_id (int4)  
Identifiant unique

pin\_id (int4)  
Clé étrangère vers personne\_info.pin\_id  
  
Lien vers personne\_info

pib\_valeur (bool)  
Valeur du champ

pib\_debut (timestampz)

Date de début pour cette valeur (si champ historisé)

pib\_fin (timestampz)

Date de fin pour cette valeur (si champ historisé)

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Identifiant de l'utilisateur ayant saisi cette valeur

### **14.2.13. public.personne\_info\_date**

Valeur d'un champ de type "Champ date" pour une personne.

pid\_id (int4)

Identifiant unique

pin\_id (int4)

Clé étrangère vers personne\_info.pin\_id

Lien vers personne\_info

pid\_valeur (date)

Valeur du champ

pid\_debut (timestampz)

Date de début pour cette valeur (si champ historisé)

pid\_fin (timestampz)

Date de fin pour cette valeur (si champ historisé)

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Identifiant de l'utilisateur ayant saisi cette valeur

### **14.2.14. public.personne\_info\_integer**

Valeur d'un champ de type "Boîtier de sélection", "Lien", "Métier", "Affectation à organisme" ou "Statut usager" pour une personne.

pii\_id (int4)

Identifiant unique

pin\_id (int4)

Clé étrangère vers personne\_info.pin\_id

Lien vers personne\_info

pii\_valeur (int4)

Valeur du champ

pii\_debut (timestampz)

Date de début pour cette valeur (si champ historisé)

pii\_fin (timestampz)

Date de fin pour cette valeur (si champ historisé)

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Identifiant de l'utilisateur ayant saisi cette valeur

#### 14.2.15. public.personne\_info\_integer2

Valeur d'un champ de type "Affectation de personnel" pour une personne.

pj\_id (int4)

Identifiant unique

pin\_id (int4)

Clé étrangère vers personne\_info.pin\_id

Lien vers personne\_info

pj\_valeur1 (int4)

Valeur du champ

pj\_valeur2 (int4)

Valeur du champ

pj\_debut (timestampz)

Date de début pour cette valeur (si champ historisé)

pj\_fin (timestampz)

Date de fin pour cette valeur (si champ historisé)

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Identifiant de l'utilisateur ayant saisi cette valeur

#### 14.2.16. public.personne\_info\_lien\_familial

Valeur d'un champ de type "Lien familial" pour une personne.

pif\_id (int4)

Identifiant unique

pin\_id (int4)

Clé étrangère vers personne\_info.pin\_id

Lien vers personne\_info

per\_id\_parent (int4)

Clé étrangère vers personne.per\_id

Identifiant de la personne parente

lfa\_id (int4)

Clé étrangère vers lien\_familial.lfa\_id

Type de lien familial

pif\_droits (varchar)

Droits du parent sur l'usager :rencontre : Rencontre médiatiséevisite : Visitesortie : Sortie et visitehebergement : Hébergement

pif\_periodicite (varchar)

Périodicité du droit (texte libre)

pif\_autorite\_parentale (int4)

Autorité du parent sur l'usager :1 : Autorité parentale2 : Tutelle3 : Aucune autorité



**14.2.17. public.personne\_info\_text**

Valeur d'un champ de type "Texte multi-ligne" pour une personne.

pit\_id (int4)

Identifiant unique

pin\_id (int4)

Clé étrangère vers personne\_info.pin\_id

Lien vers personne\_info

pit\_valeur (text)

Valeur du champ

pit\_debut (timestampz)

Date de début pour cette valeur (si champ historisé)

pit\_fin (timestampz)

Date de fin pour cette valeur (si champ historisé)

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Identifiant de l'utilisateur ayant saisi cette valeur

**14.2.18. public.personne\_info\_varchar**

Valeur d'un champ de type "Champ Texte" pour une personne.

piv\_id (int4)

Identifiant unique

pin\_id (int4)

Clé étrangère vers personne\_info.pin\_id

Lien vers personne\_info

piv\_valeur (varchar)

Valeur du champ

piv\_debut (timestampz)

Date de début pour cette valeur (si champ historisé)

piv\_fin (timestampz)

Date de fin pour cette valeur (si champ historisé)

uti\_id (int4)

Clé étrangère vers utilisateur.uti\_id

Identifiant de l'utilisateur ayant saisi cette valeur

**14.2.19. public.pgprocedures\_stats**

cal\_function\_name (varchar)

cal\_nargs (int4)

cal\_ncalls (int4)

**14.2.20. public.secteur\_infos**

sei\_id (int4)

sec\_id (int4)

Clé étrangère vers secteur.sec\_id

sec\_editable (bool)

## **14.3. Types**

**14.3.1. public.contact\_recherche**

**14.3.2. public.etablissement\_liste\_details**

**14.3.3. public.famille\_recherche**

**14.3.4. public.groupe\_liste**

**14.3.5. public.groupe\_liste\_details**

**14.3.6. public.integer2**

**14.3.7. public.personne\_cherche**

**14.3.8. public.personne\_contact\_liste**

**14.3.9. public.personne\_info\_boolean\_histo**

**14.3.10. public.personne\_info\_contact\_histo**

**14.3.11. public.personne\_info\_date\_histo**

**14.3.12. public.personne\_info\_integer\_get\_multiple\_details**

**14.3.13. public.personne\_info\_integer\_histo**

**14.3.14. public.personne\_info\_varchar\_histo**

**14.3.15. public.pgprocedures\_search\_arguments**

**14.3.16. public.pgprocedures\_search\_function**

**14.3.17. public.prise\_en\_charge\_select**

### **14.3.18. public.groupe\_cherche**

## **14.4. Fonctions**

## Name

adresse\_get — Retourne les détails de l'adresse d'un établissement/partenaire.

## Synopsis

```
adresse adresse_get(prm_token, prm_adr_id);
```

```
int4 prm_token;
```

```
int4 prm_adr_id;
```

## Description

Retourne les détails de l'adresse d'un établissement/partenaire.

Entrées :

- prm\_token : Token d'authentification
- prm\_adr\_id : Identifiant de l'adresse

## Source

```
DECLARE
    ret adresse;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM adresse WHERE adr_id = prm_adr_id;
    RETURN ret;
END;
```

## Name

armor

## Synopsis

```
text armor();
```

```
bytea ;
```

## Source

```
pg_armor
```

---

**Name**

concat2

**Synopsis**

```
text concat2(, arg1);
```

```
text ;  
text arg1;
```

**Source**

```
SELECT CASE WHEN $1 IS NULL OR $1 = '' THEN $2  
           WHEN $2 IS NULL OR $2 = '' THEN $1  
           ELSE $1 || ', ' || $2  
           END;
```

## Name

concatenate

## Synopsis

```
text concatenate();
```

```
text ;
```

## Source

```
aggregate_dummy
```

---

## Name

contact\_recherche — Retourne la liste des usagers dont une personne (personnel ou contact) est le contact.

## Synopsis

```
setof contact_recherche contact_recherche(prm_token, prm_per_id);
```

```
int4 prm_token;  
int4 prm_per_id;
```

## Description

Retourne la liste des usagers dont une personne (personnel ou contact) est le contact.

Entrées :

- prm\_token : Token d'authentification
- prm\_per\_id : Identifiant de la personne (personnel ou contact)

Retour :

- per\_id : Identifiant de l'utilisateur
- per\_libelle : Nom et prénom de l'utilisateur
- Nom du lien entre l'utilisateur et la personne (personnel ou contact)

## Source

```
DECLARE  
  row contact_recherche;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    select per_id, personne_get_libelle(prm_token, per_id), inf_libelle FROM personne_info_integer  
    inner join personne_info using(pin_id)  
    inner join meta.info using(inf_code)  
    inner join meta.infos_type USING(int_id)  
    where int_code = 'contact' AND pii_valeur = prm_per_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```



**Name**

crypt

**Synopsis**

```
text crypt(, arg1);
```

```
text ;  
text arg1;
```

**Source**

```
pg_crypt
```

---

## Name

dearmor

## Synopsis

```
bytea dearmor();
```

```
text ;
```

## Source

```
pg_dearmor
```

---

**Name**

decrypt

**Synopsis**

```
bytea decrypt(, arg1, arg2);
```

```
bytea ;  
bytea arg1;  
text arg2;
```

**Source**

```
pg_decrypt
```

---

**Name**

decrypt\_iv

**Synopsis**

```
bytea decrypt_iv(, arg1, arg2, arg3);
```

```
bytea ;  
bytea arg1;  
bytea arg2;  
text arg3;
```

**Source**

```
pg_decrypt_iv
```

---

## Name

digest

## Synopsis

```
bytea digest(, arg1);
```

```
text ;  
text arg1;
```

## Source

```
pg_digest
```

---

## Name

digest

## Synopsis

```
bytea digest(, arg1);
```

```
bytea ;  
text arg1;
```

## Source

```
pg_digest
```

---

**Name**

encrypt

**Synopsis**

```
bytea encrypt(, arg1, arg2);
```

```
bytea ;  
bytea arg1;  
text arg2;
```

**Source**

```
pg_encrypt
```

---

**Name**

encrypt\_iv

**Synopsis**

```
bytea encrypt_iv(, arg1, arg2, arg3);
```

```
bytea ;  
bytea arg1;  
bytea arg2;  
text arg3;
```

**Source**

```
pg_encrypt_iv
```

---



## Name

etablissement\_add — Ajoute un établissement ou partenaire.

## Synopsis

```
int4 etablissement_add(prm_token, prm_nom, prm_cat_id, prm_adr_adresse,  
prm_adr_cp, prm_adr_ville, prm_adr_tel, prm_adr_fax, prm_adr_email,  
prm_adr_web);
```

```
int4 prm_token;  
varchar prm_nom;  
int4 prm_cat_id;  
varchar prm_adr_adresse;  
varchar prm_adr_cp;  
varchar prm_adr_ville;  
varchar prm_adr_tel;  
varchar prm_adr_fax;  
varchar prm_adr_email;  
varchar prm_adr_web;
```

## Description

Ajoute un établissement ou partenaire.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_nom* : Nom de l'établissement/du partenaire
- *prm\_cat\_id* : Identifiant de la catégorie pour un établissement, ou NULL pour un partenaire
- *prm\_adr\_adresse* : Ligne d'adresse
- *prm\_adr\_cp* : Code postal
- *prm\_adr\_ville* : Ville
- *prm\_adr\_tel* : Téléphone
- *prm\_adr\_fax* : Fax
- *prm\_adr\_email* : Email de contact
- *prm\_adr\_web* : Site web

Retour :

- Identifiant de l'établissement/partenaire créé.

Remarques :

- Ne nécessite pas de droit particulier.

## Source

```
DECLARE  
  ret integer;  
  adr integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
```

```
INSERT INTO adresse (adr_adresse, adr_cp, adr_ville, adr_tel, adr_fax, adr_email, adr_web)
VALUES (prm_adr_adresse, prm_adr_cp, prm_adr_ville, prm_adr_tel, prm_adr_fax, prm_adr_email, prm_adr_
INSERT INTO etablissement (eta_nom, cat_id, adr_id) VALUES (prm_nom, prm_cat_id, adr) RETURNING eta_id
RETURN ret;
END;
```

## Name

etablissement\_cherche — Recherche un établissement par son nom.

## Synopsis

```
etablissement etablissement_cherche(prm_token, prm_nom);
```

```
int4 prm_token;  
varchar prm_nom;
```

## Description

Recherche un établissement par son nom.

Entrée :

- *prm\_token* : Token d'authentification
- *prm\_nom* : Nom exact de l'établissement à rechercher

## Source

```
DECLARE  
    ret etablissement;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT * INTO ret FROM etablissement WHERE eta_nom = prm_nom LIMIT 1;  
    RETURN ret;  
END;
```

## Name

`etablissement_dans_secteur_editable_liste` — Retourne la liste des établissements/partenaires d'un secteur donné auxquels les utilisateurs peuvent rajouter des groupes.

## Synopsis

```
setof etablissement etablissement_dans_secteur_editable_liste(prm_token,  
prm_interne, prm_sec_id);
```

```
int4 prm_token;  
bool prm_interne;  
int4 prm_sec_id;
```

## Description

Retourne la liste des établissements/partenaires d'un secteur donné auxquels les utilisateurs peuvent rajouter des groupes.

Entrées :

- `prm_token` : Token d'authentification
- `prm_interne` : TRUE : établissements uniquement, FALSE : partenaires uniquement, NULL : établissements et partenaires
- `prm_sec_id` : Identifiant du secteur (non NULL)

Remarques :

- Tous les partenaires sont éditables, alors qu'il est possible d'indiquer si un établissement est éditable pour un secteur donné avec la table `etablissement_secteur_edit`.

## Source

```
DECLARE
  row etablissement;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT DISTINCT etablissement.* FROM etablissement
    INNER JOIN etablissement_secteur USING(eta_id)
    LEFT JOIN etablissement_secteur_edit ON etablissement.eta_id = etablissement_secteur_edit.eta_id AND
    WHERE
      (etablissement.cat_id ISNULL OR etablissement_secteur_edit.ese_id NOTNULL)
      AND etablissement_secteur.sec_id = prm_sec_id
      AND prm_interne ISNULL OR (prm_interne AND cat_id NOTNULL) OR (NOT prm_interne AND cat_id ISNULL)
    ORDER BY eta_nom
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

etablissement\_dans\_secteur\_liste — Retourne la liste des établissements/partenaires d'un secteur donné.

## Synopsis

```
setof etablissement etablissement_dans_secteur_liste(prm_token, prm_interne,  
prm_sec_id);
```

```
int4 prm_token;  
bool prm_interne;  
int4 prm_sec_id;
```

## Description

Retourne la liste des établissements/partenaires d'un secteur donné.

- prm\_token : Token d'authentification
- prm\_interne : TRUE : établissements uniquement, FALSE : partenaires uniquement, NULL : établissements et partenaires
- prm\_sec\_id : Identifiant du secteur (non NULL)

## Source

```
DECLARE  
  row etablissement;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT etablissement.* FROM etablissement  
      INNER JOIN etablissement_secteur USING(eta_id)  
    WHERE  
      sec_id = prm_sec_id  
      AND (prm_interne ISNULL OR (prm_interne AND cat_id NOTNULL) OR (NOT prm_interne AND cat_id ISNULL))  
    ORDER BY eta_nom  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`etablissement_get` — Retourne les informations sur un établissement.

## Synopsis

```
etablissement etablissement_get(prm_token, prm_eta_id);
```

```
int4 prm_token;
```

```
int4 prm_eta_id;
```

## Description

Retourne les informations sur un établissement.

Entrées :

- `prm_token` : Token d'authentification
- `prm_eta_id` : Identifiant de l'établissement

## Source

```
DECLARE
    ret etablissement;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM etablissement WHERE eta_id = prm_eta_id;
    RETURN ret;
END;
```

## Name

etablissement\_liste — Retourne la liste des établissements et/ou partenaires.

## Synopsis

```
setof      etablissement      etablissement_liste(prm_token,      prm_interne,
prm_secteur);
```

```
int4  prm_token;
bool  prm_interne;
varchar  prm_secteur;
```

## Description

Retourne la liste des établissements et/ou partenaires.

Entrées :

- prm\_token : Token d'authentification
- prm\_interne : TRUE pour les établissements, FALSE pour les partenaires, NULL pour tous
- prm\_secteur : Code d'un secteur ou NULL. Retourne uniquement les établissements couvrant ce secteur

## Source

```
DECLARE
  row etablissement;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT DISTINCT etablissement.* FROM etablissement
    INNER JOIN etablissement_secteur USING(eta_id)
    INNER JOIN meta.secteur USING(sec_id)
  WHERE
    (prm_interne ISNULL OR (prm_interne AND cat_id NOTNULL) OR (NOT prm_interne AND cat_id ISNULL))
    AND (prm_secteur ISNULL OR prm_secteur = secteur.sec_code)
    ORDER BY eta_nom
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

etablissement\_liste\_details — Détail des établissements/partenaires couvrant un certain rôle et/ou besoin.

## Synopsis

```
setof etablissement_liste_details etablissement_liste_details(prm_token,
prm_sec_id_role, prm_sec_id_besoin, prm_interne_seuls);
```

```
int4 prm_token;
int4 prm_sec_id_role;
int4 prm_sec_id_besoin;
bool prm_interne_seuls;
```

## Description

Détail des établissements/partenaires couvrant un certain rôle et/ou besoin.

Entrées :

- `prm_token` : Token d'authentification
- `prm_sec_id_role` : Identifiant du rôle ou NULL pour ne pas filtrer sur les rôles
- `prm_sec_id_besoin` : Identifiant du besoin ou NULL pour ne pas filtrer sur les besoins
- `prm_interne_seuls` : TRUE pour retourner les établissements uniquement, FALSE pour retourner établissements et partenaires

Retour :

- `eta_id` : Identifiant de l'établissement/partenaire
- `eta_nom` : Nom
- `roles` : liste des rôles couverts par l'établissement/partenaire
- `besoins` : liste des besoins couverts par l'établissement/partenaire

Remarques :

- Nécessite les droits à la configuration "Établissement"

## Source

```
DECLARE
row etablissement_liste_details;
BEGIN
PERFORM login._token_assert (prm_token, TRUE, FALSE);
FOR row IN
SELECT eta_id, eta_nom,
(SELECT concatenate (sec_nom) FROM etablissement_secteur_liste(prm_token, eta_id, true)),
(SELECT concatenate (sec_nom) FROM etablissement_secteur_liste(prm_token, eta_id, false))
FROM etablissement
LEFT JOIN etablissement_secteur etablissement_secteur_roles USING(eta_id)
LEFT JOIN etablissement_secteur etablissement_secteur_besoins USING(eta_id)
LEFT JOIN meta.secteur roles ON roles.sec_id = etablissement_secteur_roles.sec_id AND roles.sec_est_p
LEFT JOIN meta.secteur besoins ON besoins.sec_id = etablissement_secteur_besoins.sec_id AND besoins.s
WHERE (prm_sec_id_role ISNULL OR prm_sec_id_role = roles.sec_id)
AND (prm_sec_id_besoin ISNULL OR prm_sec_id_besoin = besoins.sec_id)
AND (prm_interne_seuls ISNULL OR (prm_interne_seuls AND etablissement.cat_id NOTNULL) OR (NOT prm_int
GROUP BY eta_id, eta_nom
LOOP
RETURN NEXT row;
```



```
END LOOP ;  
END ;
```

## Name

etablissement\_liste\_details — Détail des établissements/partenaires couvrant un certain secteur.

## Synopsis

```
setof etablissement_liste_details  etablissement_liste_details(prm_token,  
prm_sec_id, prm_interne_seuls);
```

```
int4 prm_token;  
int4 prm_sec_id;  
bool prm_interne_seuls;
```

## Description

Détail des établissements/partenaires couvrant un certain secteur.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_sec\_id* : Identifiant du secteur ou NULL pour ne pas filtrer sur les secteurs
- *prm\_interne\_seuls* : TRUE pour retourner les établissements uniquement, FALSE pour retourner établissements et partenaires

Retour :

- *eta\_id* : Identifiant de l'établissement/partenaire
- *eta\_nom* : Nom
- *roles* : liste des rôles couverts par l'établissement/partenaire
- *besoins* : liste des besoins couverts par l'établissement/partenaire

Remarques :

- Nécessite les droits à la configuration "Établissement" ou "Réseau"

## Source

```
DECLARE
  row etablissement_liste_details;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, TRUE);
  FOR row IN
    SELECT eta_id, eta_nom, concatenate (roles.sec_nom), concatenate (besoins.sec_nom)
    FROM etablissement
    LEFT JOIN etablissement_secteur USING(eta_id)
    LEFT JOIN meta.secteur roles ON roles.sec_id = etablissement_secteur.sec_id AND roles.sec_est_prise_e
    LEFT JOIN meta.secteur besoins ON besoins.sec_id = etablissement_secteur.sec_id AND besoins.sec_est_p
    WHERE (prm_sec_id ISNULL OR prm_sec_id = roles.sec_id OR prm_sec_id = besoins.sec_id)
    AND (prm_interne_seuls ISNULL OR (prm_interne_seuls AND etablissement.cat_id NOTNULL) OR (NOT prm_int
    GROUP BY eta_id, eta_nom
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

`etablissement_liste_par_secteur` — Retourne la liste des établissements/partenaires en filtrant sur les secteurs couverts par les groupes des établissements/partenaires.

## Synopsis

```
setof etablissement etablissement_liste_par_secteur(prm_token, prm_secteur);
```

```
int4 prm_token;
```

```
varchar prm_secteur;
```

## Description

Retourne la liste des établissements/partenaires en filtrant sur les secteurs couverts par les groupes des établissements/partenaires.

Entrées :

- `prm_token` : Token d'authentification
- `prm_secteur` : Code secteur ou NULL. Retourne uniquement les établissements dont au moins un groupe couvre le secteur donné

## Source

```
DECLARE
    row etablissement;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT etablissement.* FROM etablissement
            INNER JOIN groupe USING(eta_id)
            LEFT JOIN groupe_secteur USING(grp_id)
            LEFT JOIN meta.secteur USING(sec_id)
            WHERE prm_secteur ISNULL OR sec_code = prm_secteur
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

`etablissement_secteur_edit_get` — Retourne TRUE si l'établissement est éditable (les utilisateurs peuvent y rajouter des groupes) pour le secteur donné, FALSE sinon.

## Synopsis

```
bool etablissement_secteur_edit_get(prm_token, prm_eta_id, prm_secteur);
```

```
int4 prm_token;
```

```
int4 prm_eta_id;
```

```
varchar prm_secteur;
```

## Description

Retourne TRUE si l'établissement est éditable (les utilisateurs peuvent y rajouter des groupes) pour le secteur donné, FALSE sinon.

Entrées :

- `prm_token` : Token d'authentification
- `prm_eta_id` : Identifiant de l'établissement
- `prm_secteur` : Code du secteur

## Source

```
DECLARE
    ret boolean;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    RETURN EXISTS (SELECT 1 FROM etablissement_secteur_edit
        INNER JOIN meta.secteur USING (sec_id)
        WHERE eta_id = prm_eta_id AND sec_code = prm_secteur);
END;
```

## Name

`etablissement_secteur_edit_liste` — Retourne la liste des secteurs pour lesquels un établissement est éditable (pour lesquels les utilisateurs peuvent rajouter des groupes).

## Synopsis

```
setof secteur etablissement_secteur_edit_liste(prm_token, prm_eta_id);
```

```
int4 prm_token;  
int4 prm_eta_id;
```

## Description

Retourne la liste des secteurs pour lesquels un établissement est éditable (pour lesquels les utilisateurs peuvent rajouter des groupes).

Entrées :

- `prm_token` : Token d'authentification
- `prm_eta_id` : Identifiant de l'établissement

Remarques :

- Nécessite les droits à la configuration "Établissement"

## Source

```
DECLARE  
    row meta.secteur;  
BEGIN  
    PERFORM login._token_assert (prm_token, TRUE, FALSE);  
    FOR row IN  
        SELECT secteur.* FROM meta.secteur  
        INNER JOIN etablissement_secteur_edit USING(sec_id)  
        WHERE eta_id = prm_eta_id  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

etablissement\_secteur\_liste — Retourne la liste des secteurs que couvre un établissement/partenaire.

## Synopsis

```
setof   secteur   etablissement_secteur_liste(prm_token,      prm_eta_id,  
prm_est_prise_en_charge);
```

```
int4   prm_token;  
int4   prm_eta_id;  
bool   prm_est_prise_en_charge;
```

## Description

Retourne la liste des secteurs que couvre un établissement/partenaire.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eta\_id* : Identifiant de l'établissement/partenaire
- *prm\_est\_prise\_en\_charge* (booléen) : TRUE pour retourner uniquement les rôles, FALSE pour les besoins, NULL pour tous

## Source

```
DECLARE  
  row meta.secteur;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT secteur.* FROM meta.secteur  
      INNER JOIN etablissement_secteur USING(sec_id)  
      WHERE eta_id = prm_eta_id AND (prm_est_prise_en_charge ISNULL OR sec_est_prise_en_charge = prm_est_pr  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`etablissement_secteurs_edit_set` — Indique pour un établissement donné pour quels secteurs il est éditable (il est possible pour les utilisateurs de rajouter des groupes).

## Synopsis

```
void etablissement_secteurs_edit_set(prm_token, prm_eta_id, prm_secteurs);
```

```
int4 prm_token;
```

```
int4 prm_eta_id;
```

```
varchar[] prm_secteurs;
```

## Description

Indique pour un établissement donné pour quels secteurs il est éditable (il est possible pour les utilisateurs de rajouter des groupes).

Entrées :

- `prm_token` : Token d'authentification
- `prm_eta_id` : Identifiant de l'établissement
- `prm_secteurs` : Tableau de codes de secteurs

Remarques :

- Nécessite les droits à la configuration "Établissement"

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM etablissement_secteur_edit WHERE eta_id = prm_eta_id;
  IF prm_secteurs NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP
      INSERT INTO etablissement_secteur_edit (eta_id, sec_id) VALUES (prm_eta_id, (SELECT sec_id FROM met
    END LOOP;
  END IF;
END;
```

## Name

etablissement\_secteurs\_set — Indique les secteurs couverts par un établissement/partenaire.

## Synopsis

```
void etablissement_secteurs_set(prm_token, prm_eta_id, prm_secteurs);
```

```
int4 prm_token;
```

```
int4 prm_eta_id;
```

```
varchar[] prm_secteurs;
```

## Description

Indique les secteurs couverts par un établissement/partenaire.

Entrées :

- *prm\_token* : Token d'authentification
- *prm\_eta\_id* : Identifiant de l'établissement/partenaire
- *prm\_secteurs* : Tableau de codes de secteurs

Remarques :

- Ne nécessite pas de droit de configuration particulier

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  DELETE FROM etablissement_secteur WHERE eta_id = prm_eta_id;
  IF prm_secteurs NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP
      INSERT INTO etablissement_secteur (eta_id, sec_id) VALUES (prm_eta_id, (SELECT sec_id FROM meta.sec
    END LOOP;
  END IF;
END;
```



## Name

etablissement\_supprime — Supprime un établissement/partenaire

## Synopsis

```
void etablissement_supprime(prm_token, prm_eta_id);
```

```
int4 prm_token;
```

```
int4 prm_eta_id;
```

## Description

Supprime un établissement/partenaire

Entrées :

- `prm_token` : Token d'authentification
- `prm_eta_id` : Identifiant de l'établissement/partenaire

Remarques :

- Nécessite les droits à la configuration "Établissement" ou "Réseau"

## Source

```
DECLARE
  adr integer;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, TRUE);
  SELECT adr_id INTO adr FROM etablissement WHERE eta_id = prm_eta_id;
  DELETE FROM etablissement_secteur WHERE eta_id = prm_eta_id;
  DELETE FROM etablissement_secteur_edit WHERE eta_id = prm_eta_id;
  DELETE FROM etablissement WHERE eta_id = prm_eta_id;
  IF adr NOTNULL THEN
    DELETE FROM adresse WHERE adr_id = adr;
  END IF;
END;
```

## Name

etablissement\_update — Modifie les informations d'un établissement/partenaire.

## Synopsis

```
void etablissement_update(prm_token, prm_eta_id, prm_nom, prm_cat_id,  
prm_adr_adresse, prm_adr_cp, prm_adr_ville, prm_adr_tel, prm_adr_fax,  
prm_adr_email, prm_adr_web);
```

```
int4 prm_token;  
int4 prm_eta_id;  
varchar prm_nom;  
int4 prm_cat_id;  
varchar prm_adr_adresse;  
varchar prm_adr_cp;  
varchar prm_adr_ville;  
varchar prm_adr_tel;  
varchar prm_adr_fax;  
varchar prm_adr_email;  
varchar prm_adr_web;
```

## Description

Modifie les informations d'un établissement/partenaire.

Entrées :

- prm\_token : Token d'authentification
- prm\_eta\_id : Identifiant de l'établissement/partenaire à modifier
- prm\_nom : Nouveau nom
- prm\_cat\_id : Identifiant de la nouvelle catégorie d'établissement ou NULL pour un partenaire
- prm\_adr\_adresse : Ligne d'adresse
- prm\_adr\_cp : Code postal
- prm\_adr\_ville : Ville
- prm\_adr\_tel : Téléphone
- prm\_adr\_fax : Fax
- prm\_adr\_email : Email de contact
- prm\_adr\_web : Site web

Nécessite les droits à la configuration "Établissement" ou "Réseau"

## Source

```
DECLARE  
  adr integer;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, TRUE);  
  UPDATE etablissement SET eta_nom = prm_nom, cat_id = prm_cat_id WHERE eta_id = prm_eta_id;  
  SELECT adr_id INTO adr FROM etablissement WHERE eta_id = prm_eta_id;  
  IF adr ISNULL THEN  
    INSERT INTO adresse (adr_adresse, adr_cp, adr_ville, adr_tel, adr_fax, adr_email, adr_web)
```

```
VALUES (prm_adr_adresse, prm_adr_cp, prm_adr_ville, prm_adr_tel, prm_adr_fax, prm_adr_email, prm_adr_web)
UPDATE etablisement SET adr_id = adr WHERE eta_id = prm_eta_id;
ELSE
UPDATE adresse SET
  adr_adresse = prm_adr_adresse,
  adr_cp = prm_adr_cp,
  adr_ville = prm_adr_ville,
  adr_tel = prm_adr_tel,
  adr_fax = prm_adr_fax,
  adr_email = prm_adr_email,
  adr_web = prm_adr_web
WHERE adr_id = adr;
END IF;
END;
```

## Name

famille\_recherche — Recherche les usagers ayant un lien familial avec une personne.

## Synopsis

```
setof famille_recherche  famille_recherche(prm_token,    prm_parent_id,
prm_inf_id);
```

```
int4 prm_token;
int4 prm_parent_id;
int4 prm_inf_id;
```

## Description

Recherche les usagers ayant un lien familial avec une personne.

Entrées :

- prm\_token : Token d'authentification
- prm\_parent\_id : Identifiant du parent
- prm\_inf\_id : Identifiant du champ de type famille sur lequel rechercher le lien familial

## Source

```
DECLARE
  row famille_recherche;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT per_id, personne_get_libelle (prm_token, per_id), lfa_id, pif_autorite_parentale, pif_droits,
    FROM personne_info_lien_familial
    INNER JOIN personne_info USING(pin_id)
    INNER JOIN meta.info USING(info_code)
    INNER JOIN meta.infos_type USING(int_id)
    WHERE int_code = 'famille' AND per_id_parent = prm_parent_id AND inf_id = prm_inf_id
    ORDER BY personne_get_libelle (prm_token, per_id)
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

**Name**

gen\_random\_bytes

**Synopsis**

```
bytea gen_random_bytes();
```

```
int4 ;
```

**Source**

```
pg_random_bytes
```

---

**Name**

gen\_salt

**Synopsis**

```
text gen_salt(, arg1);
```

```
text ;  
int4 arg1;
```

**Source**

```
pg_gen_salt_rounds
```

---

## Name

gen\_salt

## Synopsis

```
text gen_salt();
```

```
text ;
```

## Source

```
pg_gen_salt
```

---

## Name

groupe\_add — Ajoute un groupe d'utilisateurs.

## Synopsis

```
int4  groupe_add(prm_token,  prm_nom,  prm_eta_id,  prm_debut,  prm_fin,  
prm_notes);
```

```
int4  prm_token;  
varchar prm_nom;  
int4  prm_eta_id;  
date  prm_debut;  
date  prm_fin;  
varchar prm_notes;
```

## Description

Ajoute un groupe d'utilisateurs.

Entrées :

- `prm_token` : Token d'authentification
- `prm_nom` : Nom du groupe
- `prm_eta_id` : Identifiant de l'établissement/partenaire
- `prm_debut` : Date de début d'existence du groupe
- `prm_fin` : Date de fin d'existence du groupe
- `prm_notes` : Notes concernant le groupe

Remarques :

- Ne nécessite pas de droit particulier.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    INSERT INTO groupe (grp_nom, eta_id, grp_debut, grp_fin, grp_notes) VALUES (prm_nom, prm_eta_id, prm_de,  
        RETURNING grp_id INTO ret;  
    RETURN ret;  
END;
```



## Name

`groupe_aide_a_la_personne_update` — Modifie les informations d'un groupe spécifiques au secteur aide à la personne.

## Synopsis

```
void groupe_aide_a_la_personne_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur aide à la personne.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_aide_a_la_personne_contact = prm_contact,  
    grp_aide_a_la_personne_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

`groupe_aide_financiere_directe_update` — Modifie les informations d'un groupe spécifiques au secteur aide financière directe.

## Synopsis

```
void      groupe_aide_financiere_directe_update(prm_token,      prm_grp_id,  
prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur aide financière directe.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_aide_financiere_directe_contact = prm_contact,  
    grp_aide_financiere_directe_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_aide\_formalites\_update — Modifie les informations d'un groupe spécifiques au secteur aide\_formalites.

## Synopsis

```
void groupe_aide_formalites_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur aide\_formalites.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_aide_formalites_contact = prm_contact,  
    grp_aide_formalites_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_cherche — Recherche le groupe d'utilisateurs d'un établissement/partenaire à partir de leurs noms.

## Synopsis

```
setof groupe_cherche groupe_cherche(prm_token, prm_eta_nom, prm_grp_nom);
```

```
int4 prm_token;
```

```
varchar prm_eta_nom;
```

```
varchar prm_grp_nom;
```

## Description

Recherche le groupe d'utilisateurs d'un établissement/partenaire à partir de leurs noms.

Entrées :

- prm\_token : Token d'authentification
- prm\_eta\_nom : Nom de l'établissement du groupe
- prm\_grp\_nom : Nom du groupe d'utilisateurs recherché

Retour :

- eta\_id : Identifiant de l'établissement/partenaire
- grp\_id : Identifiant du groupe

Remarques :

- La recherche est insensible à la casse.

## Source

```
DECLARE
    row groupe_cherche;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT eta_id, grp_id FROM etablisement INNER JOIN groupe USING(eta_id)
            WHERE eta_nom ilike prm_eta_nom AND grp_nom ilike prm_grp_nom
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

groupe\_culture\_update — Modifie les informations d'un groupe spécifiques au secteur culture.

## Synopsis

```
void groupe_culture_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur culture.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_culture_contact = prm_contact,  
    grp_culture_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_decideur\_update — Modifie les informations d'un groupe spécifiques au secteur décideur.

## Synopsis

```
void groupe_decideur_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur décideur.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_decideur_contact = prm_contact,  
    grp_decideur_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_dietetique\_update — Modifie les informations d'un groupe spécifiques au secteur dietetique.

## Synopsis

```
void groupe_dietetique_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur dietetique.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_dietetique_contact = prm_contact,  
    grp_dietetique_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_divertissement\_update — Modifie les informations d'un groupe spécifiques au secteur divertissement.

## Synopsis

```
void groupe_divertissement_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur divertissement.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_divertissement_contact = prm_contact,  
    grp_divertissement_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```



## Name

groupe\_droits\_de\_sejour\_update — Modifie les informations d'un groupe spécifiques au secteur droits\_de\_sejour.

## Synopsis

```
void groupe_droits_de_sejour_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur droits\_de\_sejour.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_droits_de_sejour_contact = prm_contact,  
    grp_droits_de_sejour_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_education\_update — Modifie les informations d'un groupe spécifiques au secteur éducation.

## Synopsis

```
void groupe_education_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur éducation.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_education_contact = prm_contact,  
    grp_education_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_emploi\_update — Modifie les informations d'un groupe spécifiques au secteur emploi.

## Synopsis

```
void groupe_emploi_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur emploi.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_emploi_contact = prm_contact,  
    grp_emploi_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_entretien\_update — Modifie les informations d'un groupe spécifiques au secteur entretien.

## Synopsis

```
void groupe_entretien_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur entretien.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_entretien_contact = prm_contact,  
    grp_entretien_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

`groupe_equipement_personnel_update` — Modifie les informations d'un groupe spécifiques au secteur équipement personnel.

## Synopsis

```
void groupe_equipement_personnel_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur équipement personnel.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_equipement_personnel_contact = prm_contact,  
    grp_equipement_personnel_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_ergotherapie\_update — Modifie les informations d'un groupe spécifiques au secteur ergotherapie.

## Synopsis

```
void    groupe_ergotherapie_update(prm_token,    prm_grp_id,    prm_contact,  
prm_type);
```

```
int4    prm_token;  
int4    prm_grp_id;  
int4    prm_contact;  
int4    prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur ergotherapie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_ergotherapie_contact = prm_contact,  
    grp_ergotherapie_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_famille\_update — Modifie les informations d'un groupe spécifiques au secteur famille.

## Synopsis

```
void groupe_famille_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur famille.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_famille_contact = prm_contact,  
    grp_famille_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_filtre — Recherche de groupes en appliquant différents filtres.

## Synopsis

```
setof groupe groupe_filtre(prm_token, prm_secteur, prm_sec_id, prm_interne,
prm_eta_id);
```

```
int4 prm_token;
varchar prm_secteur;
int4 prm_sec_id;
bool prm_interne;
int4 prm_eta_id;
```

## Description

Recherche de groupes en appliquant différents filtres.

Entrées :

- prm\_token : token d'authentification
- prm\_secteur : Code d'un secteur, pour rechercher les groupes couvrant un secteur particulier
- prm\_sec\_id : Identifiant d'un secteur, pour rechercher les groupes couvrant un secteur particulier
- prm\_interne : NULL = établissements/partenaires, TRUE = établissements seuls, FALSE = partenaires seuls
- prm\_eta\_id : Identifiant d'un établissement/partenaire, pour rechercher parmi les groupes de cet établissement/partenaire

## Source

```
DECLARE
  row groupe;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT DISTINCT groupe.* FROM groupe
    LEFT JOIN groupe_secteur USING(grp_id)
    LEFT JOIN meta.secteur USING(sec_id)
    LEFT JOIN etablissement USING(eta_id)
    WHERE (prm_secteur ISNULL OR sec_code = prm_secteur)
    AND (prm_sec_id ISNULL OR prm_sec_id = sec_id)
    AND (prm_interne ISNULL OR (prm_interne AND cat_id NOTNULL) OR (NOT prm_interne AND cat_id ISNULL))
    AND (prm_eta_id ISNULL OR groupe.eta_id = prm_eta_id)
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```



## Name

groupe\_financeur\_update — Modifie les informations d'un groupe spécifiques au secteur financeur.

## Synopsis

```
void groupe_financeur_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur financeur.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_financeur_contact = prm_contact,  
    grp_financeur_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

`groupe_get` — Retourne les informations sur un groupe d'utilisateurs.

## Synopsis

```
groupe groupe_get(prm_token, prm_grp_id);
```

```
int4 prm_token;
```

```
int4 prm_grp_id;
```

## Description

Retourne les informations sur un groupe d'utilisateurs.

Entrées :

- `prm_token` : Token d'authentification
- `prm_grp_id` : Identifiant du groupe

## Source

```
DECLARE
    ret groupe;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM groupe WHERE grp_id = prm_grp_id;
    RETURN ret;
END;
```

## Name

groupe\_hebergement\_update — Modifie les informations d'un groupe spécifiques au secteur hébergement.

## Synopsis

```
void groupe_hebergement_update(prm_token, prm_grp_id, prm_adresse, prm_cp,
prm_ville, prm_contact, prm_type);
```

```
int4 prm_token;
int4 prm_grp_id;
varchar prm_adresse;
varchar prm_cp;
varchar prm_ville;
int4 prm_contact;
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur hébergement.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  UPDATE groupe SET
    grp_hebergement_adresse = prm_adresse,
    grp_hebergement_cp = prm_cp,
    grp_hebergement_ville = prm_ville,
    grp_hebergement_contact = prm_contact,
    grp_hebergement_type = prm_type
  WHERE grp_id = prm_grp_id;
END;
```

## Name

`groupe_info_secteur_get` — Retourne les informations indiquant sur quel champ de fiche usager est faite l'affectation d'un usager à un groupe pour un secteur donné.

## Synopsis

```
setof groupe_info_secteur groupe_info_secteur_get(prm_token, prm_grp_id,  
prm_sec_code);
```

```
int4 prm_token;  
int4 prm_grp_id;  
varchar prm_sec_code;
```

## Description

Retourne les informations indiquant sur quel champ de fiche usager est faite l'affectation d'un usager à un groupe pour un secteur donné.

Entrées :

- `prm_token` : Token d'authentification
- `prm_grp_id` : Identifiant du groupe ou NULL
- `prm_sec_code` : Code du secteur ou NULL

## Source

```
DECLARE  
  row groupe_info_secteur;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT groupe_info_secteur.* FROM groupe_info_secteur  
    LEFT JOIN meta.secteur USING(sec_id)  
    WHERE (prm_grp_id ISNULL OR prm_grp_id = grp_id) AND  
    (prm_sec_code ISNULL OR prm_sec_code = sec_code)  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

`groupe_info_secteur_save` — Indique le champ de fiche usager sur lequel est faite l'affectation d'un usager à un groupe pour un secteur donné.

## Synopsis

```
void    groupe_info_secteur_save(prm_token,    prm_grp_id,    prm_sec_code,  
prm_inf_id);
```

```
int4    prm_token;  
int4    prm_grp_id;  
varchar prm_sec_code;  
int4    prm_inf_id;
```

## Description

Indique le champ de fiche usager sur lequel est faite l'affectation d'un usager à un groupe pour un secteur donné.

Entrées :

- `prm_token` : Token d'authentification
- `prm_grp_id` : Identifiant du groupe
- `prm_sec_code` : Code du secteur
- `prm_inf_id` : Identifiant du champ de saisie de fiche usager

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    UPDATE groupe_info_secteur SET inf_id = prm_inf_id WHERE grp_id = prm_grp_id AND sec_id = (SELECT sec_id  
    IF NOT FOUND THEN  
        INSERT INTO groupe_info_secteur (grp_id, sec_id, inf_id) VALUES (prm_grp_id, (SELECT sec_id FROM meta  
    END IF;  
END;
```

## Name

`groupe_info_secteurs_set` — Indique le champ de fiche usager sur lequel est faite l'affectation d'un usager à un groupe pour une liste de secteurs.

## Synopsis

```
void    groupe_info_secteurs_set(prm_token,    prm_grp_id,    prm_inf_id,  
prm_secteurs);
```

```
int4    prm_token;  
int4    prm_grp_id;  
int4    prm_inf_id;  
varchar[] prm_secteurs;
```

## Description

Indique le champ de fiche usager sur lequel est faite l'affectation d'un usager à un groupe pour une liste de secteurs.

Entrées :

- `prm_token` : Token d'authentification
- `prm_grp_id` : Identifiant du groupe
- `prm_inf_id` : Identifiant du champ de saisie de fiche usager
- `prm_secteurs` : Tableau de codes du secteur

## Source

```
DECLARE  
    row meta.secteur;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN SELECT * FROM meta.secteur LOOP  
        PERFORM groupe_info_secteur_save (prm_token, prm_grp_id, row.sec_code, NULL);  
    END LOOP;  
    IF prm_secteurs NOTNULL THEN  
        FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP  
            PERFORM groupe_info_secteur_save (prm_token, prm_grp_id, prm_secteurs[i], prm_inf_id);  
        END LOOP;  
    END IF;  
END;
```

## Name

groupe\_justice\_update — Modifie les informations d'un groupe spécifiques au secteur justice.

## Synopsis

```
void groupe_justice_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur justice.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_justice_contact = prm_contact,  
    grp_justice_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_kinesitherapie\_update — Modifie les informations d'un groupe spécifiques au secteur kinesitherapie.

## Synopsis

```
void groupe_kinesitherapie_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur kinesitherapie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_kinesitherapie_contact = prm_contact,  
    grp_kinesitherapie_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```



## Name

`groupe_liste_details` — Retourne le détail des groupes d'un établissement/partenaire donné couvrant un rôle/besoin donné.

## Synopsis

```
setof groupe_liste_details groupe_liste_details(prm_token, prm_eta_id,  
prm_sec_id_role, prm_sec_id_besoin, prm_interne_seuls);
```

```
int4 prm_token;  
int4 prm_eta_id;  
int4 prm_sec_id_role;  
int4 prm_sec_id_besoin;  
bool prm_interne_seuls;
```

## Description

Retourne le détail des groupes d'un établissement/partenaire donné couvrant un rôle/besoin donné.

Entrées :

- `prm_token` : Token d'authentification
- `prm_eta_id` : Retourne uniquement les groupes d'un établissement/partenaire, NULL pour retourner les groupes de tous les établissements/partenaires
- `prm_sec_id_role` : Retourne uniquement les groupes couvrant le rôle indiqué, NULL pour ne pas filtrer sur les rôles
- `prm_sec_id_besoin` : Retourne uniquement les groupes couvrant le besoin indiqué, NULL pour ne pas filtrer sur les besoins
- `prm_interne_seuls` : TRUE pour retourner uniquement les groupes des établissements, NULL pour retourner les groupes des établissements et partenaires

Retour :

- `grp_id` : Identifiant du groupe
- `eta_nom` : Nom de l'établissement
- `grp_nom` : Nom du groupe
- `roles` : Liste des noms de rôles couverts par le groupe
- `besoins` : Liste des noms de besoins couverts par le groupe
- `grp_debut` : Date de début d'activité du groupe
- `grp_fin` : Date de fin d'activité du groupe

## Source

```
DECLARE  
  row groupe_liste_details;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT grp_id, eta_nom, grp_nom,  
           (SELECT concatenate (sec_nom) FROM groupe_secteur_liste (prm_token, grp_id, true)),  
           (SELECT concatenate (sec_nom) FROM groupe_secteur_liste (prm_token, grp_id, false)),  
           grp_debut, grp_fin
```

```
FROM groupe
LEFT JOIN etablissement USING(eta_id)
LEFT JOIN groupe_secteur groupe_secteur_role USING(grp_id)
LEFT JOIN groupe_secteur groupe_secteur_besoin USING(grp_id)
LEFT JOIN meta.secteur roles ON roles.sec_id = groupe_secteur_role.sec_id AND roles.sec_est_prise_en_
LEFT JOIN meta.secteur besoins ON besoins.sec_id = groupe_secteur_besoin.sec_id AND NOT besoins.sec_e
WHERE (prm_sec_id_role ISNULL OR prm_sec_id_role = roles.sec_id) AND (prm_sec_id_besoin ISNULL OR prm
AND (NOT prm_interne_seuls OR cat_id NOTNULL)
AND (prm_eta_id ISNULL OR groupe.eta_id = prm_eta_id)
GROUP BY grp_id, eta_nom, grp_nom, grp_debut, grp_fin
LOOP
RETURN NEXT row;
END LOOP;
END;
```

## Name

groupe\_orthophonie\_update — Modifie les informations d'un groupe spécifiques au secteur orthophonie.

## Synopsis

```
void    groupe_orthophonie_update(prm_token,    prm_grp_id,    prm_contact,  
prm_type);
```

```
int4    prm_token;  
int4    prm_grp_id;  
int4    prm_contact;  
int4    prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur orthophonie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_orthophonie_contact = prm_contact,  
    grp_orthophonie_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_pedagogie\_update — Modifie les informations d'un groupe spécifiques au secteur pédagogie.

## Synopsis

```
void groupe_pedagogie_update(prm_token, prm_grp_id, prm_type, prm_niveau,  
prm_contact);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_type;  
int4 prm_niveau;  
int4 prm_contact;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur pédagogie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_pedagogie_type = prm_type,  
    grp_pedagogie_niveau = prm_niveau,  
    grp_pedagogie_contact = prm_contact  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_physiotherapie\_update — Modifie les informations d'un groupe spécifiques au secteur physiotherapie.

## Synopsis

```
void groupe_physiotherapie_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur physiotherapie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_physiotherapie_contact = prm_contact,  
    grp_physiotherapie_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_prise\_en\_charge\_update — Modifie les informations d'un groupe spécifiques au secteur prise en charge.

## Synopsis

```
void groupe_prise_en_charge_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur prise en charge.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_prise_en_charge_contact = prm_contact,  
    grp_prise_en_charge_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_projet\_update — Modifie les informations d'un groupe spécifiques au secteur projet.

## Synopsis

```
void groupe_projet_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur projet.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_projet_contact = prm_contact,  
    grp_projet_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_protection\_juridique\_update — Modifie les informations d'un groupe spécifiques au secteur protection juridique.

## Synopsis

```
void groupe_protection_juridique_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur protection juridique.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_protection_juridique_contact = prm_contact,  
    grp_protection_juridique_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```



## Name

groupe\_psychologie\_update — Modifie les informations d'un groupe spécifiques au secteur psychologie.

## Synopsis

```
void    groupe_psychologie_update(prm_token,    prm_grp_id,    prm_contact,  
prm_type);
```

```
int4    prm_token;  
int4    prm_grp_id;  
int4    prm_contact;  
int4    prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur psychologie.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_psychologie_contact = prm_contact,  
    grp_psychologie_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_psychomotricite\_update — Modifie les informations d'un groupe spécifiques au secteur psychomotricite.

## Synopsis

```
void groupe_psychomotricite_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur psychomotricite.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_psychomotricite_contact = prm_contact,  
    grp_psychomotricite_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_restaurations\_update — Modifie les informations d'un groupe spécifiques au secteur restauration.

## Synopsis

```
void    groupe_restaurations_update(prm_token,    prm_grp_id,    prm_contact,  
prm_type);
```

```
int4    prm_token;  
int4    prm_grp_id;  
int4    prm_contact;  
int4    prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur restauration.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_restaurations_contact = prm_contact,  
    grp_restaurations_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_sante\_update — Modifie les informations d'un groupe spécifiques au secteur santé.

## Synopsis

```
void groupe_sante_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur santé.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_sante_contact = prm_contact,  
    grp_sante_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_secteur\_liste — Retourne la liste des secteurs couverts par un groupe.

## Synopsis

```
setof      secteur      groupe_secteur_liste(prm_token,      prm_grp_id,  
prm_est_prise_en_charge);
```

```
int4  prm_token;  
int4  prm_grp_id;  
bool  prm_est_prise_en_charge;
```

## Description

Retourne la liste des secteurs couverts par un groupe.

Entrées :

- prm\_token : Token d'authentification
- prm\_grp\_id : Identifiant du groupe
- prm\_est\_prise\_en\_charge : NULL pour lister tous les secteurs, TRUE pour les secteurs de prise en charge ou FALSE les autres secteurs

## Source

```
DECLARE  
  row meta.secteur;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT secteur.* FROM meta.secteur  
    INNER JOIN groupe_secteur USING(sec_id)  
    WHERE grp_id = prm_grp_id  
    AND (prm_est_prise_en_charge ISNULL OR sec_est_prise_en_charge = prm_est_prise_en_charge)  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

groupe\_secteurs\_set — Indique les secteurs couverts par un groupe.

## Synopsis

```
void groupe_secteurs_set(prm_token, prm_grp_id, prm_secteurs);
```

```
int4 prm_token;  
int4 prm_grp_id;  
varchar[] prm_secteurs;
```

## Description

Indique les secteurs couverts par un groupe.

Entrées :

- `prm_token` : Token d'authentification
- `prm_grp_id` : Identifiant du groupe
- `prm_secteurs` : Tableau d'identifiants de secteurs

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  DELETE FROM groupe_secteur WHERE grp_id = prm_grp_id;  
  IF prm_secteurs NOTNULL THEN  
    FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP  
      INSERT INTO groupe_secteur (grp_id, sec_id) VALUES (prm_grp_id, (SELECT sec_id FROM meta.secteur WH  
    END LOOP;  
  END IF;  
END;
```

## Name

groupe\_sejour\_update — Modifie les informations d'un groupe spécifiques au secteur séjour.

## Synopsis

```
void groupe_sejour_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur séjour.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_sejour_contact = prm_contact,  
    grp_sejour_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_social\_update — Modifie les informations d'un groupe spécifiques au secteur social.

## Synopsis

```
void groupe_social_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur social.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_social_contact = prm_contact,  
    grp_social_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```



## Name

groupe\_soins\_infirmiers\_update — Modifie les informations d'un groupe spécifiques au secteur soins\_infirmiers.

## Synopsis

```
void groupe_soins_infirmiers_update(prm_token, prm_grp_id, prm_contact,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur soins\_infirmiers.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_soins_infirmiers_contact = prm_contact,  
    grp_soins_infirmiers_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_sport\_update — Modifie les informations d'un groupe spécifiques au secteur sport.

## Synopsis

```
void groupe_sport_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur sport.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_sport_contact = prm_contact,  
    grp_sport_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_supprime — Supprime un groupe d'utilisateurs.

## Synopsis

```
void groupe_supprime(prm_token, prm_grp_id);
```

```
int4 prm_token;
```

```
int4 prm_grp_id;
```

## Description

Supprime un groupe d'utilisateurs.

Entrées :

- prm\_token : Token d'authentification
- prm\_grp\_id : Identifiant du groupe

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  DELETE FROM groupe_info_secteur WHERE grp_id = prm_grp_id;
  DELETE FROM groupe_secteur WHERE grp_id = prm_grp_id;
  DELETE FROM groupe WHERE grp_id = prm_grp_id;
END;
```

## Name

groupe\_transport\_update — Modifie les informations d'un groupe spécifiques au secteur transport.

## Synopsis

```
void groupe_transport_update(prm_token, prm_grp_id, prm_contact, prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
int4 prm_contact;  
int4 prm_type;
```

## Description

Modifie les informations d'un groupe spécifiques au secteur transport.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET  
    grp_transport_contact = prm_contact,  
    grp_transport_type = prm_type  
  WHERE grp_id = prm_grp_id;  
END;
```

## Name

groupe\_type\_secteur\_update — Modifie le type d'un groupe pour un secteur particulier

## Synopsis

```
void groupe_type_secteur_update(prm_token,    prm_grp_id,    prm_sec_code,  
prm_type);
```

```
int4 prm_token;  
int4 prm_grp_id;  
varchar prm_sec_code;  
int4 prm_type;
```

## Description

Modifie le type d'un groupe pour un secteur particulier

## Source

```
DECLARE  
    sql varchar;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    sql = 'UPDATE public.groupe SET grp_' || prm_sec_code || '_type = ' || prm_type || ' WHERE grp_id = ' ||  
    EXECUTE sql;  
END;
```

## Name

groupe\_update — Modifie les informations d'un groupe d'usagers.

## Synopsis

```
bool groupe_update(prm_token, prm_grp_id, prm_nom, prm_eta_id, prm_debut,  
prm_fin, prm_notes);
```

```
int4 prm_token;  
int4 prm_grp_id;  
varchar prm_nom;  
int4 prm_eta_id;  
date prm_debut;  
date prm_fin;  
varchar prm_notes;
```

## Description

Modifie les informations d'un groupe d'usagers.

Entrées :

- `prm_token` : Token d'authentification
- `prm_grp_id` : Identifiant du groupe à mettre à jour
- `prm_nom` : Nouveau nom du groupe
- `prm_eta_id` : Identifiant de l'établissement/partenaire auquel appartient le groupe
- `prm_debut` : Date de début d'existence du groupe
- `prm_fin` : Date de fin d'existence du groupe
- `prm_notes` : Notes concernant le groupe

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  UPDATE groupe SET grp_nom = prm_nom, eta_id = prm_eta_id, grp_debut = prm_debut, grp_fin = prm_fin, grp  
    WHERE grp_id = prm_grp_id;  
  RETURN FOUND;  
END;
```

**Name**

hmac

**Synopsis**

```
bytea hmac(, arg1, arg2);
```

```
text ;  
text arg1;  
text arg2;
```

**Source**

```
pg_hmac
```

---

**Name**

hmac

**Synopsis**

```
bytea hmac(, arg1, arg2);
```

```
bytea ;  
bytea arg1;  
text arg2;
```

**Source**

```
pg_hmac
```

---



## Name

`periods_overlap` — Retourne TRUE si 2 périodes de temps se chevauchent, FALSE sinon.

## Synopsis

```
bool periods_overlap(du1, au1, du2, au2);
```

date *du1*;

date *au1*;

date *du2*;

date *au2*;

## Description

Retourne TRUE si 2 périodes de temps se chevauchent, FALSE sinon.

## Source

```
DECLARE
  ret BOOLEAN;
BEGIN
  -- RAISE NOTICE '% % % %', du1, au1, du2, au2;
  SELECT (COALESCE (du1, timestamp '-INFINITY'), COALESCE (au1, timestamp 'INFINITY'))
         OVERLAPS (COALESCE (du2, timestamp '-INFINITY'), COALESCE (au2, timestamp 'INFINITY')) INTO ret;
  -- RAISE NOTICE '%', ret;
  RETURN ret;
END;
```

## Name

personne\_ajoute — Ajoute une nouvelle personne.

## Synopsis

```
int4 personne_ajoute(prm_token, prm_ent_code);
```

```
int4 prm_token;  
varchar prm_ent_code;
```

## Description

Ajoute une nouvelle personne.

## Source

```
DECLARE  
    ret integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    INSERT INTO personne (ent_code) VALUES (prm_ent_code)  
        RETURNING per_id INTO ret;  
    RETURN ret;  
END;
```

## Name

personne\_cherche

## Synopsis

```
setof personne_cherche personne_cherche(prm_token, prm_nom, prm_prenom);
```

```
int4 prm_token;
```

```
varchar prm_nom;
```

```
varchar prm_prenom;
```

## Source

```
DECLARE
    row personne_cherche;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT per_id,
            personne_info_varchar_get(prm_token, per_id, 'nom') || ' ' || personne_info_varchar_get(prm_token, pe
        FROM personne
        WHERE (prm_nom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'nom') ilike prm_nom || '%')
            AND (prm_prenom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'prenom') ilike prm_prenom ||
            ORDER BY personne_info_varchar_get(prm_token, per_id, 'nom') || ' ' || personne_info_varchar_get(prm
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

personne\_cherche

## Synopsis

```
setof personne_cherche personne_cherche(prm_token, prm_nom, prm_prenom,
prm_type, prm_grp_id);
```

```
int4 prm_token;
varchar prm_nom;
varchar prm_prenom;
varchar prm_type;
int4 prm_grp_id;
```

## Source

```

DECLARE
    row personne_cherche;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT per_id,
            COALESCE (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ', '') || COALESCE (personne_info_
            (CASE WHEN prm_type = 'usager' THEN ''
                WHEN prm_type = 'famille' THEN ''
                WHEN prm_type = 'contact' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id =
                WHEN prm_type = 'personnel' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id
            END)
        FROM personne
        LEFT JOIN personne_groupe USING(per_id)
        WHERE (prm_nom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'nom') ilike prm_nom || '%')
            AND (prm_prenom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'prenom') ilike prm_prenom ||
            AND ent_code = prm_type
            AND (prm_grp_id ISNULL OR (personne_groupe.grp_id = prm_grp_id AND CURRENT_TIMESTAMP BETWEEN COALESC
            ORDER BY COALESCE (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ', '') || COALESCE (pers
            (CASE WHEN prm_type = 'usager' THEN ''
                WHEN prm_type = 'famille' THEN ''
                WHEN prm_type = 'contact' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id =
                WHEN prm_type = 'personnel' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id
            END)
        LOOP
            RETURN NEXT row;
        END LOOP;
END;
```

## Name

personne\_cherche

## Synopsis

```
setof personne_cherche personne_cherche(prm_token, prm_nom, prm_prenom,
prm_type);
```

```
int4 prm_token;
varchar prm_nom;
varchar prm_prenom;
varchar prm_type;
```

## Source

```

DECLARE
    row personne_cherche;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT per_id,
            COALESCE (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ', '') || COALESCE (personne_info_
            (CASE WHEN prm_type = 'usager' THEN ''
                WHEN prm_type = 'famille' THEN ''
                WHEN prm_type = 'contact' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id =
                WHEN prm_type = 'personnel' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id
            END)
        FROM personne
    WHERE (prm_nom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'nom') ilike prm_nom || '%')
        AND (prm_prenom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'prenom') ilike prm_prenom ||
        AND ent_code = prm_type
    ORDER BY COALESCE (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ', '') || COALESCE (pers
    (CASE WHEN prm_type = 'usager' THEN ''
        WHEN prm_type = 'famille' THEN ''
        WHEN prm_type = 'contact' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id =
        WHEN prm_type = 'personnel' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id
    END)
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

personne\_cherche2

## Synopsis

```
setof personne_cherche personne_cherche2(prm_token, prm_nom, prm_prenom,
prm_type, prm_secteur);
```

```
int4 prm_token;
varchar prm_nom;
varchar prm_prenom;
varchar prm_type;
varchar prm_secteur;
```

## Source

```

DECLARE
    row personne_cherche;
BEGIN
-- RAISE WARNING '%', prm_secteur;
    PERFORM login.token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT DISTINCT per_id,
            COALESCE (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ', '') || COALESCE (personne_info_
            (CASE WHEN prm_type = 'usager' THEN ''
                WHEN prm_type = 'famille' THEN ''
                WHEN prm_type = 'contact' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id =
                WHEN prm_type = 'personnel' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id
            END)
        FROM personne
        WHERE (prm_nom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'nom') ilike prm_nom || '%')
            AND (prm_prenom ISNULL OR personne_info_varchar_get(prm_token, per_id, 'prenom') ilike prm_prenom ||
            AND ent_code = prm_type
            AND (prm_secteur ISNULL OR
                personne_info_integer_get (prm_token, per_id, 'personnel_metier') IN (SELECT met_id FROM meta.metie
                personne_info_integer_get (prm_token, per_id, 'contact_metier') IN (SELECT met_id FROM meta.metier
            )
            ORDER BY COALESCE (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ', '') || COALESCE (pers
            (CASE WHEN prm_type = 'usager' THEN ''
                WHEN prm_type = 'famille' THEN ''
                WHEN prm_type = 'contact' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id =
                WHEN prm_type = 'personnel' THEN COALESCE (' (' || (SELECT met_nom FROM meta.metier WHERE met_id
            END)
    LOOP
        RETURN NEXT row;
    END LOOP;
END;
```

## Name

personne\_cherche\_exact

## Synopsis

```
setof int4 personne_cherche_exact(prm_token, prm_nom, prm_prenom, prm_type);
```

```
int4 prm_token;  
varchar prm_nom;  
varchar prm_prenom;  
varchar prm_type;
```

## Source

```
DECLARE  
  row RECORD;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT per_id FROM personne  
    WHERE personne_info_varchar_get(prm_token, per_id, 'nom') ilike prm_nom  
    AND COALESCE (personne_info_varchar_get(prm_token, per_id, 'prenom'), '') ilike COALESCE (prm_prenom, '')  
    AND ent_code = prm_type  
  LOOP  
    RETURN NEXT row.per_id;  
  END LOOP;  
END;
```

## Name

personne\_cherche\_exact\_tout

## Synopsis

```
setof int4 personne_cherche_exact_tout(prm_token, prm_nom_prenom, prm_type);
```

```
int4 prm_token;
```

```
varchar prm_nom_prenom;
```

```
varchar prm_type;
```

## Source

```
DECLARE
    row RECORD;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT per_id FROM personne
        WHERE TRIM (personne_info_varchar_get(prm_token, per_id, 'nom') || ' ' || COALESCE (personne_info_var
        AND ent_code = prm_type
    LOOP
        RETURN NEXT row.per_id;
    END LOOP;
END;
```



## Name

`personne_contact_liste` — Retourne la liste des contacts ayant un métier dans un secteur donné affectés à un établissement donné.

## Synopsis

```
setof personne_contact_liste personne_contact_liste(prm_token, prm_filtre,  
prm_secteur, prm_eta_id);
```

```
int4 prm_token;  
varchar prm_filtre;  
varchar prm_secteur;  
int4 prm_eta_id;
```

## Description

Retourne la liste des contacts ayant un métier dans un secteur donné affectés à un établissement donné.

## Source

```
DECLARE
  row1 RECORD;
  int_per_id integer;
  row personne_contact_liste%ROWTYPE;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row1 IN
    SELECT per_id FROM personne
    WHERE (prm_filtre ISNULL OR ent_code = prm_filtre)
    AND personne_info_integer_get(prm_token, per_id, ent_code || '_metier')
    IN (SELECT met_id FROM meta.metier_secteur INNER JOIN meta.secteur USING(sec_id) WHERE sec_code = p
    AND (prm_eta_id ISNULL
        OR (((SELECT inf_multiple FROM meta.info WHERE inf_code = ent_code || '_affectation') = FALSE
            AND (SELECT valeur1 FROM personne_info_integer2_get(prm_token, per_id, ent_code
            OR ((SELECT inf_multiple FROM meta.info WHERE inf_code = ent_code || '_affectati
            AND prm_eta_id IN (SELECT valeur1 FROM personne_info_integer2_get_multiple(prm_
    LOOP
      row.per_id = row1.per_id;
      SELECT personne_info_varchar_get (prm_token, row1.per_id, 'nom') || ' ' || personne_info_varchar_get
      RETURN NEXT row;
    END LOOP;
END;
```

## Name

personne\_etablissement\_update

## Synopsis

```
void personne_etablissement_update(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Source

```
DECLARE
    eta RECORD;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    DELETE FROM personne_etablissement WHERE per_id = prm_per_id;
    FOR eta IN SELECT eta_id FROM etablissement LOOP
        IF EXISTS (SELECT 1 FROM personne_groupe INNER JOIN groupe USING(grp_id) WHERE per_id = prm_per_id AND eta_id = eta_id) THEN
            INSERT INTO personne_etablissement (per_id, eta_id) VALUES (prm_per_id, eta.eta_id);
        END IF;
    END LOOP;
END;
```

## Name

personne\_get — Retourne les informations sur une personne.

## Synopsis

```
personne personne_get(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Description

Retourne les informations sur une personne.

## Source

```
DECLARE
  ret personne;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT * INTO ret FROM personne WHERE per_id = prm_per_id;
  RETURN ret;
END;
```

## Name

personne\_get\_libelle — Retourne le prénom suivi du nom d'une personne.

## Synopsis

```
varchar personne_get_libelle(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Description

Retourne le prénom suivi du nom d'une personne.

## Source

```
DECLARE
  ret VARCHAR;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT COALESCE (personne_info_varchar_get(prm_token, per_id, 'prenom'), '') || ' ' || COALESCE (person
  RETURN ret;
END;
```

## Name

`personne_get_libelle_initiale` — Retourne l'initiale du prénom suivi du nom d'une personne.

## Synopsis

```
vvarchar personne_get_libelle_initiale(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Description

Retourne l'initiale du prénom suivi du nom d'une personne.

## Source

```
DECLARE
  ret VARCHAR;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT COALESCE (SUBSTRING(personne_info_vvarchar_get(prm_token, per_id, 'prenom') FROM 1 FOR 1) || '.',
  RETURN ret;
END;
```

## Name

personne\_groupe\_ajoute — Affecte un usager à un groupe.

## Synopsis

```
int4      personne_groupe_ajoute(prm_token,      prm_per_id,      prm_grp_id,
prm_debut, prm_fin,  prm_notes,  prm_cycle_statut,  prm_cycle_date_demande,
prm_cycle_date_demande_renouvellement,      prm_hebergement_chambre,
prm__decideur_financeur);
```

```
int4 prm_token;
int4 prm_per_id;
int4 prm_grp_id;
date prm_debut;
date prm_fin;
text prm_notes;
int4 prm_cycle_statut;
date prm_cycle_date_demande;
date prm_cycle_date_demande_renouvellement;
varchar prm_hebergement_chambre;
int4 prm__decideur_financeur;
```

## Description

Affecte un usager à un groupe.

## Source

```
DECLARE
    ret integer;
    tmp integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    INSERT INTO personne_groupe (per_id, grp_id, peg_debut, peg_fin, peg_cycle_statut, peg_cycle_date_demande,
    VALUES (prm_per_id, prm_grp_id, prm_debut, prm_fin, prm_cycle_statut, prm_cycle_date_demande, prm_cycle
        RETURNING peg_id INTO ret;
    EXECUTE personne_etablissement_update (prm_token, prm_per_id);
    SELECT * INTO tmp FROM meta.meta_statut_usager_calcule ('statut_usager', prm_per_id);
    RETURN ret;
END;
```

## Name

personne\_groupe\_info — Retourne les informations sur l'affectation d'un usager à un groupe.

## Synopsis

```
groupe_liste personne_groupe_info(prm_token, prm_peg_id);
```

```
int4 prm_token;  
int4 prm_peg_id;
```

## Description

Retourne les informations sur l'affectation d'un usager à un groupe.

## Source

```
DECLARE  
  row groupe_liste%ROWTYPE;  
BEGIN  
  PERFORM login.token_assert (prm_token, FALSE, FALSE);  
  select  
    personne_groupe.peg_id,  
    groupe.grp_id,  
    groupe.eta_id,  
    etablisement.eta_nom,  
    groupe.grp_nom,  
    personne_groupe.peg_debut,  
    personne_groupe.peg_fin,  
    personne_groupe.peg_notes,  
    personne_groupe.peg_cycle_statut,  
    personne_groupe.peg_cycle_date_demande,  
    personne_groupe.peg_cycle_date_demande_renouvellement,  
    personne_groupe.peg_hebergement_chambre,  
    personne_groupe.peg__decideur_financeur  
  INTO row  
  FROM groupe  
  LEFT JOIN etablisement USING(eta_id)  
  INNER JOIN groupe_secteur USING(grp_id)  
  INNER JOIN meta.secteur USING(sec_id)  
  INNER JOIN personne_groupe USING (grp_id)  
  where peg_id = prm_peg_id ;  
  RETURN row;  
END;
```

## Name

`personne_groupe_liste2` — Retourne la liste des affectations d'une personne à des groupes, associées à un champ groupe donné.

## Synopsis

```
setof  groupe_liste      personne_groupe_liste2(prm_token,      prm_per_id,
prm_inf_id);
```

```
int4  prm_token;
int4  prm_per_id;
int4  prm_inf_id;
```

## Description

Retourne la liste des affectations d'une personne à des groupes, associées à un champ groupe donné.

## Source

```
DECLARE
  row groupe_liste%ROWTYPE;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT DISTINCT
      personne_groupe.peg_id,
      groupe.grp_id,
      groupe.eta_id,
      etablisement.eta_nom,
      groupe.grp_nom,
      personne_groupe.peg_debut,
      personne_groupe.peg_fin,
      personne_groupe.peg_notes,
      personne_groupe.peg_cycle_statut,
      personne_groupe.peg_cycle_date_demande,
      personne_groupe.peg_cycle_date_demande_renouvellement,
      personne_groupe.peg_hebergement_chambre,
      personne_groupe.peg__decideur_financeur
    FROM groupe
    INNER JOIN personne_groupe USING (grp_id)
    INNER JOIN groupe_secteur USING (grp_id)
    INNER JOIN meta.secteur ON secteur.sec_id = groupe_secteur.sec_id
    INNER JOIN meta.info ON info.inf__groupe_type = secteur.sec_code
    INNER JOIN groupe_info_secteur ON groupe_info_secteur.grp_id=groupe.grp_id AND groupe_info_secteur.in
    LEFT JOIN etablisement USING(eta_id)
    where per_id = prm_per_id AND groupe_info_secteur.inf_id = prm_inf_id
    ORDER BY peg_debut DESC, peg_fin DESC
  LOOP
    return NEXT row;
  END LOOP;
END;
```



## Name

personne\_groupe\_supprime — Supprime l'affectation d'un usager à un groupe.

## Synopsis

```
void personne_groupe_supprime(prm_token, prm_peg_id);
```

```
int4 prm_token;  
int4 prm_peg_id;
```

## Description

Supprime l'affectation d'un usager à un groupe.

## Source

```
DECLARE  
    tmp integer;  
    per integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT per_id INTO per FROM personne_groupe WHERE peg_id = prm_peg_id;  
    DELETE FROM personne_groupe WHERE peg_id = prm_peg_id;  
    EXECUTE personne_etablissement_update (prm_token, per);  
    SELECT * INTO tmp FROM meta.meta_statut_usager_calcule ('statut_usager', per);  
END;
```

## Name

personne\_groupe\_update — Modifie les informations d'affectation d'un usager à un groupe.

## Synopsis

```
void    personne_groupe_update(prm_token,      prm_peg_id,      prm_grp_id,
prm_debut, prm_fin, prm_notes, prm_cycle_statut, prm_cycle_date_demande,
prm_cycle_date_demande_renouvellement,      prm__hebergement_chambre,
prm__decideur_financeur);
```

```
int4    prm_token;
int4    prm_peg_id;
int4    prm_grp_id;
date    prm_debut;
date    prm_fin;
text    prm_notes;
int4    prm_cycle_statut;
date    prm_cycle_date_demande;
date    prm_cycle_date_demande_renouvellement;
varchar prm__hebergement_chambre;
int4    prm__decideur_financeur;
```

## Description

Modifie les informations d'affectation d'un usager à un groupe.

## Source

```
DECLARE
    tmp integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    UPDATE personne_groupe SET
        grp_id = prm_grp_id,
        peg_debut = prm_debut,
        peg_fin = prm_fin,
        peg_notes = prm_notes,
        peg_cycle_statut = prm_cycle_statut,
        peg_cycle_date_demande = prm_cycle_date_demande,
        peg_cycle_date_demande_renouvellement = prm_cycle_date_demande_renouvellement,
        peg__hebergement_chambre = prm__hebergement_chambre ,
        peg__decideur_financeur = prm__decideur_financeur
    WHERE peg_id = prm_peg_id;
    SELECT * INTO tmp FROM meta.meta_statut_usager_calcule ('statut_usager', (SELECT per_id FROM personne
END;
```

## Name

personne\_info\_boolean\_get

## Synopsis

```
bool personne_info_boolean_get(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE
    ret boolean;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT pib_valeur INTO ret
        FROM personne_info_boolean
        INNER JOIN personne_info USING (pin_id)
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        ORDER BY pib_id DESC LIMIT 1;
    RETURN ret;
END;
```

## Name

personne\_info\_boolean\_get\_histo

## Synopsis

```
setof                               personne_info_boolean_histo
personne_info_boolean_get_histo(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
```

## Source

```
DECLARE
  row personne_info_boolean_histo;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT pib_debut, pib_fin,
           CASE WHEN pib_valeur THEN 'oui' else 'non' END,
           login.utilisateur_prenon_nom(prm_token, uti_id)
    FROM personne_info_boolean
    INNER JOIN personne_info USING (pin_id)
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code
    ORDER BY pib_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

personne\_info\_boolean\_set

## Synopsis

```
void    personne_info_boolean_set(prm_token,    prm_per_id,    prm_inf_code,
prm_valeur, prm_util_id);
```

```
int4    prm_token;
int4    prm_per_id;
varchar prm_inf_code;
bool    prm_valeur;
int4    prm_util_id;
```

## Source

```
DECLARE
    new_pin_id integer;
    historique boolean;
    pib_id_dernier integer;
    the_pin_id integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT inf_historique INTO historique FROM meta.info WHERE inf_code = prm_inf_code;
    IF historique THEN
        -- Valeur historisée :
        SELECT pin_id INTO the_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code;
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
        END IF;
        SELECT pib_id INTO pib_id_dernier FROM personne_info_boolean WHERE pin_id = the_pin_id
            ORDER BY pib_id DESC LIMIT 1;
        -- on met à jour seulement si la valeur n'existe pas encore ou est différente
        IF prm_valeur IS DISTINCT FROM (SELECT pib_valeur FROM personne_info_boolean WHERE pib_id = pib_id_dernier)
            -- On met la date de fin à la précédente valeur
            UPDATE personne_info_boolean SET pib_fin = CURRENT_TIMESTAMP WHERE pib_id = pib_id_dernier;
            -- puis on crée la nouvelle valeur
            INSERT INTO personne_info_boolean (pin_id, pib_valeur, pib_debut, uti_id) VALUES (the_pin_id, prm_valeur,
            END IF;
    ELSE
        -- Valeur ni historisée ni multiple : on écrase l'ancienne valeur
        UPDATE personne_info_boolean
            SET pib_valeur = prm_valeur, uti_id = prm_util_id WHERE pib_debut ISNULL AND pib_fin ISNULL AND
            pib_valeur IS DISTINCT FROM prm_valeur AND
            pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code);
        IF NOT FOUND AND NOT EXISTS (SELECT 1 FROM personne_info_boolean WHERE
            pib_debut ISNULL AND pib_fin ISNULL AND
            pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code)) THEN
            -- ou on la crée si elle n'existe pas
            SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code;
            IF NOT FOUND THEN
                INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
            END IF;
            INSERT INTO personne_info_boolean (pin_id, pib_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_util_id);
        END IF;
    END IF;
END;
```

## Name

personne\_info\_contact\_get\_histo

## Synopsis

```
setof                               personne_info_contact_histo
personne_info_contact_get_histo(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
```

## Source

```
DECLARE
  row personne_info_contact_histo;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT pii_debut, pii_fin,
           personne_info_varchar_get (prm_token, pii_valeur, 'nom') || ' ' || personne_info_varchar_get (
           login.utilisateur_prenon_nom(prm_token, uti_id)
    FROM personne_info_integer
    INNER JOIN personne_info USING (pin_id)
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code
    ORDER BY pii_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

personne\_info\_date\_get

## Synopsis

```
date personne_info_date_get(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE
    ret date;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT pid_valeur INTO ret
        FROM personne_info_date
        INNER JOIN personne_info USING (pin_id)
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        ORDER BY pid_id DESC LIMIT 1;
    RETURN ret;
END;
```

## Name

personne\_info\_date\_get\_histo

## Synopsis

```
setof  personne_info_date_histo  personne_info_date_get_histo(prm_token,  
prm_per_id, prm_inf_code);
```

```
int4  prm_token;  
int4  prm_per_id;  
varchar  prm_inf_code;
```

## Source

```
DECLARE  
  row  personne_info_date_histo;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT pid_debut, pid_fin, pid_valeur,  
           login.utilisateur_prenon_nom(prm_token, uti_id)  
    FROM  personne_info_date  
    INNER JOIN  personne_info USING (pin_id)  
    WHERE  per_id = prm_per_id AND inf_code = prm_inf_code  
    ORDER BY pid_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```



## Name

personne\_info\_date\_get\_multiple

## Synopsis

```
setof date personne_info_date_get_multiple(prm_token, prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE  
    row RECORD;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    FOR row IN  
        SELECT pid_valeur  
            FROM personne_info_date  
            INNER JOIN personne_info USING (pin_id)  
            WHERE per_id = prm_per_id AND inf_code = prm_inf_code  
            ORDER BY pid_id  
    LOOP  
        RETURN NEXT row.pid_valeur;  
    END LOOP;  
END;
```

## Name

personne\_info\_date\_prepare\_multiple

## Synopsis

```
void      personne_info_date_prepare_multiple(prm_token,      prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    DELETE FROM personne_info_date WHERE pin_id = (SELECT pin_id FROM personne_info WHERE per_id = prm_per_  
END;
```

## Name

personne\_info\_date\_set

## Synopsis

```
void personne_info_date_set(prm_token, prm_per_id, prm_inf_code, prm_valeur,
prm_uti_id);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
date prm_valeur;
int4 prm_uti_id;
```

## Source

```

DECLARE
    new_pin_id integer;
    historique boolean;
    multiple boolean;
    pid_id_dernier integer;
    the_pin_id integer;
BEGIN
    PERFORM login_token_assert (prm_token, FALSE, FALSE);
    SELECT inf_historique INTO historique FROM meta.info WHERE inf_code = prm_inf_code;
    SELECT inf_multiple INTO multiple FROM meta.info WHERE inf_code = prm_inf_code;
    IF historique THEN
        -- Valeur historisée :
        SELECT pin_id INTO the_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO the_pin_id;
        END IF;
        SELECT pid_id INTO pid_id_dernier FROM personne_info_date WHERE pin_id = the_pin_id
        ORDER BY pid_id DESC LIMIT 1;
        -- on met à jour seulement si la valeur n'existe pas encore ou est différente
        IF prm_valeur IS DISTINCT FROM (SELECT pid_valeur FROM personne_info_date WHERE pid_id = pid_id_dernier) THEN
            -- On met la date de fin à la précédente valeur
            UPDATE personne_info_date SET pid_fin = CURRENT_TIMESTAMP WHERE pid_id = pid_id_dernier;
            -- puis on crée la nouvelle valeur
            INSERT INTO personne_info_date (pin_id, pid_valeur, pid_debut, uti_id) VALUES (the_pin_id, prm_valeur,
            pid_debut, prm_uti_id) RETURNING pin_id INTO new_pin_id;
        END IF;
    ELSIF multiple THEN
        SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
        END IF;
        INSERT INTO personne_info_date (pin_id, pid_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_uti_id) RETURNING pin_id INTO new_pin_id;
    ELSE
        -- Valeur ni historisée ni multiple : on écrase l'ancienne valeur
        UPDATE personne_info_date
        SET pid_valeur = prm_valeur, uti_id = prm_uti_id WHERE pid_debut ISNULL AND pid_fin ISNULL AND
        pid_valeur IS DISTINCT FROM prm_valeur AND
        pin_id = (SELECT pin_id FROM personne_info
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code);
        IF NOT FOUND AND NOT EXISTS (SELECT 1 FROM personne_info_date WHERE pid_debut ISNULL AND pid_fin ISNULL AND
        pin_id = (SELECT pin_id FROM personne_info
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code)) THEN
            -- ou on la crée si elle n'existe pas
            SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
            IF NOT FOUND THEN
                INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
            END IF;
            INSERT INTO personne_info_date (pin_id, pid_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_uti_id) RETURNING pin_id INTO new_pin_id;
        END IF;
    END IF;
END;

```

## Name

personne\_info\_integer2\_delete

## Synopsis

```
void personne_info_integer2_delete(prm_token, prm_pij_id);
```

```
int4 prm_token;  
int4 prm_pij_id;
```

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    DELETE FROM personne_info_integer2 WHERE pij_id = prm_pij_id;  
END;
```

## Name

personne\_info\_integer2\_get

## Synopsis

```
integer2 personne_info_integer2_get(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE
    ret integer2;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT pij_valeur1, pij_valeur2, pij_id INTO ret
        FROM personne_info_integer2
        INNER JOIN personne_info USING (pin_id)
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        ORDER BY pij_id DESC LIMIT 1;
    RETURN ret;
END;
```

## Name

personne\_info\_integer2\_get\_multiple

## Synopsis

```
setof integer2 personne_info_integer2_get_multiple(prm_token, prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE  
  row RECORD;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT pij_valeur1, pij_valeur2, pij_id  
    FROM personne_info_integer2  
    INNER JOIN personne_info USING (pin_id)  
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code  
    ORDER BY pij_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

personne\_info\_integer2\_get\_par\_id

## Synopsis

```
integer2 personne_info_integer2_get_par_id(prm_token, prm_pij_id);
```

```
int4 prm_token;
```

```
int4 prm_pij_id;
```

## Source

```
DECLARE
  ret integer2;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT pij_valeur1, pij_valeur2, pij_id INTO ret
     FROM personne_info_integer2
     WHERE pij_id = prm_pij_id;
  RETURN ret;
END;
```

## Name

personne\_info\_integer2\_prepare\_multiple

## Synopsis

```
void    personne_info_integer2_prepare_multiple(prm_token,      prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    DELETE FROM personne_info_integer2 WHERE pin_id = (SELECT pin_id FROM personne_info WHERE per_id = prm_per_id);  
END;
```



## Name

personne\_info\_integer2\_set

## Synopsis

```
int4 personne_info_integer2_set(prm_token, prm_per_id, prm_inf_code,
prm_valeur1, prm_valeur2, prm_util_id);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
int4 prm_valeur1;
int4 prm_valeur2;
int4 prm_util_id;
```

## Source

```

DECLARE
    new_pin_id integer;
    multiple boolean;
    pii_id_dernier integer;
    the_pin_id integer;
    ret integer;
BEGIN
    PERFORM login_token_assert (prm_token, FALSE, FALSE);
    SELECT inf_multiple INTO multiple FROM meta.info WHERE inf_code = prm_inf_code;
    IF multiple THEN
        SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code;
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
        END IF;
        INSERT INTO personne_info_integer2 (pin_id, pij_valeur1, pij_valeur2, uti_id) VALUES (new_pin_id, prm_valeur1, prm_valeur2, prm_util_id)
            RETURNING pij_id INTO ret;
        RETURN ret;
    ELSE
        -- Valeur ni historisée ni multiple : on écrase l'ancienne valeur
        UPDATE personne_info_integer2
            SET pij_valeur1 = prm_valeur1, pij_valeur2 = prm_valeur2, uti_id = prm_util_id WHERE pij_debut ISNULL
            (pij_valeur1 IS DISTINCT FROM prm_valeur1 OR pij_valeur2 IS DISTINCT FROM prm_valeur2) AND
            pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code);
        IF NOT FOUND AND NOT EXISTS (SELECT 1 FROM personne_info_integer2 WHERE pij_debut ISNULL AND pij_fin ISNULL
            AND (pij_valeur1 IS DISTINCT FROM prm_valeur1 OR pij_valeur2 IS DISTINCT FROM prm_valeur2)
            AND pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code)) THEN
            -- ou on la crée si elle n'existe pas
            SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code;
            IF NOT FOUND THEN
                INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
            END IF;
            INSERT INTO personne_info_integer2 (pin_id, pij_valeur1, pij_valeur2, uti_id) VALUES (new_pin_id, prm_valeur1, prm_valeur2, prm_util_id)
                RETURNING pij_id INTO ret;
            RETURN ret;
        ELSE
            RETURN 0;
        END IF;
    END IF;
END;
```

## Name

personne\_info\_integer\_delete

## Synopsis

```
void personne_info_integer_delete(prm_token, prm_pii_id);
```

```
int4 prm_token;
```

```
int4 prm_pii_id;
```

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  DELETE FROM personne_info_integer WHERE pii_id = prm_pii_id;
END;
```

## Name

personne\_info\_integer\_get

## Synopsis

```
int4 personne_info_integer_get(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  SELECT pii_valeur INTO ret
  FROM personne_info_integer
  INNER JOIN personne_info USING (pin_id)
  WHERE per_id = prm_per_id AND inf_code = prm_inf_code
  ORDER BY pii_id DESC LIMIT 1;
  RETURN ret;
END;
```

## Name

personne\_info\_integer\_get\_histo

## Synopsis

```
setof                               personne_info_integer_histo
personne_info_integer_get_histo(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
```

## Source

```
DECLARE
  row personne_info_integer_histo;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT pii_debut, pii_fin, sen_libelle,
           login.utilisateur_prenon_nom(prm_token, uti_id)
    FROM personne_info_integer
    INNER JOIN personne_info USING (pin_id)
    INNER JOIN meta.selection_entree ON pii_valeur = sen_id
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code
    ORDER BY pii_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

personne\_info\_integer\_get\_multiple

## Synopsis

```
setof int4 personne_info_integer_get_multiple(prm_token, prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE  
  row RECORD;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT pii_valeur  
    FROM personne_info_integer  
    INNER JOIN personne_info USING (pin_id)  
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code  
    ORDER BY pii_id  
  LOOP  
    RETURN NEXT row.pii_valeur;  
  END LOOP;  
END;
```

## Name

personne\_info\_integer\_get\_multiple\_details

## Synopsis

```
setof                               personne_info_integer_get_multiple_details
personne_info_integer_get_multiple_details(prm_token,           prm_per_id,
prm_inf_code);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
```

## Source

```
DECLARE
  row personne_info_integer_get_multiple_details;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT pii_id, pii_valeur
    FROM personne_info_integer
    INNER JOIN personne_info USING (pin_id)
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code
    ORDER BY pii_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

personne\_info\_integer\_prepare\_multiple

## Synopsis

```
void      personne_info_integer_prepare_multiple(prm_token,      prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    DELETE FROM personne_info_integer WHERE pin_id = (SELECT pin_id FROM personne_info WHERE per_id = prm_p  
END;
```

## Name

personne\_info\_integer\_set

## Synopsis

```
int4    personne_info_integer_set(prm_token,    prm_per_id,    prm_inf_code,
prm_valeur, prm_util_id);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
int4 prm_valeur;
int4 prm_util_id;
```

## Source

```
DECLARE
    new_pin_id integer;
    historique boolean;
    multiple boolean;
    pii_id_dernier integer;
    the_pin_id integer;
    ret integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    ret = 0;
-- RAISE NOTICE '% % %', prm_per_id, prm_inf_code, prm_valeur;
    SELECT inf_historique INTO historique FROM meta.info WHERE inf_code = prm_inf_code;
    SELECT inf_multiple INTO multiple FROM meta.info WHERE inf_code = prm_inf_code;
    IF historique THEN
-- Valeur historisée :
        SELECT pin_id INTO the_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO ret;
        END IF;
        SELECT pii_id INTO pii_id_dernier FROM personne_info_integer WHERE pin_id = the_pin_id
        ORDER BY pii_id DESC LIMIT 1;
-- on met à jour seulement si la valeur n'existe pas encore ou est différente
        IF prm_valeur IS DISTINCT FROM (SELECT pii_valeur FROM personne_info_integer WHERE pii_id = pii_id_dernier)
-- On met la date de fin à la précédente valeur
            UPDATE personne_info_integer SET pii_fin = CURRENT_TIMESTAMP WHERE pii_id = pii_id_dernier;
-- puis on crée la nouvelle valeur
            INSERT INTO personne_info_integer (pin_id, pii_valeur, pii_debut, uti_id) VALUES (the_pin_id, prm_valeur,
                RETURNING pii_id INTO ret;
        END IF;
    ELSIF multiple THEN
        SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO ret;
        END IF;
        INSERT INTO personne_info_integer (pin_id, pii_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_util_id)
        RETURNING pii_id INTO ret;
    ELSE
-- Valeur ni historisée ni multiple : on écrase l'ancienne valeur
        UPDATE personne_info_integer
        SET pii_valeur = prm_valeur, uti_id = prm_util_id WHERE pii_debut ISNULL AND pii_fin ISNULL AND
        pii_valeur IS DISTINCT FROM prm_valeur AND
        pin_id = (SELECT pin_id FROM personne_info
            WHERE per_id = prm_per_id AND inf_code = prm_inf_code);
        IF NOT FOUND AND NOT EXISTS (SELECT 1 FROM personne_info_integer WHERE pii_debut ISNULL AND pii_fin ISNULL AND
            pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code)) THEN
-- ou on la crée si elle n'existe pas
            SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
            IF NOT FOUND THEN
                INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO ret;
```



```
END IF;  
INSERT INTO personne_info_integer (pin_id, pii_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_  
RETURNING pii_id INTO ret;  
END IF;  
END IF;  
RETURN ret;  
END;
```

## Name

personne\_info\_lien\_familial\_delete

## Synopsis

```
void personne_info_lien_familial_delete(prm_token, prm_pif_id);
```

```
int4 prm_token;
```

```
int4 prm_pif_id;
```

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  DELETE FROM personne_info_lien_familial WHERE pif_id = prm_pif_id;
END;
```

## Name

personne\_info\_lien\_familial\_get\_multiple

## Synopsis

```
setof                               personne_info_lien_familial
personne_info_lien_familial_get_multiple(prm_token,           prm_per_id,
prm_inf_code);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
```

## Source

```
DECLARE
  row RECORD;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT personne_info_lien_familial.*
    FROM personne_info_lien_familial
    INNER JOIN personne_info USING (pin_id)
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code
    ORDER BY pif_id
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

personne\_info\_lien\_familial\_get\_par\_id

## Synopsis

```
personne_info_lien_familial  
personne_info_lien_familial_get_par_id(prm_token, prm_pif_id);
```

```
int4 prm_token;  
int4 prm_pif_id;
```

## Source

```
DECLARE  
  ret personne_info_lien_familial;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  SELECT personne_info_lien_familial.* INTO ret  
    FROM personne_info_lien_familial  
    WHERE pif_id = prm_pif_id;  
  RETURN ret;  
END;
```

## Name

personne\_info\_lien\_familial\_set

## Synopsis

```
int4 personne_info_lien_familial_set(prm_token, prm_per_id, prm_inf_code,
prm_per_id_parent, prm_lfa_id, prm_autorite_parentale, prm_droits,
prm_periodicite);
```

```
int4 prm_token;
int4 prm_per_id;
varchar prm_inf_code;
int4 prm_per_id_parent;
int4 prm_lfa_id;
int4 prm_autorite_parentale;
varchar prm_droits;
varchar prm_periodicite;
```

## Source

```
DECLARE
    new_pin_id integer;
    ret integer;
BEGIN
    PERFORM login_token_assert (prm_token, FALSE, FALSE);
    -- On considère que c'est multiple
    SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code;
    IF NOT FOUND THEN
        INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO :
    END IF;
    INSERT INTO personne_info_lien_familial (pin_id, per_id_parent, lfa_id, pif_autorite_parentale, pif_dro
        VALUES (new_pin_id, prm_per_id_parent, prm_lfa_id, prm_autorite_parentale, prm_droits, prm_periodic
        RETURNING pif_id INTO ret;
    RETURN ret;
END;
```

## Name

personne\_info\_text\_get

## Synopsis

```
text personne_info_text_get(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE
    ret text;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT pit_valeur INTO ret
        FROM personne_info_text
        INNER JOIN personne_info USING (pin_id)
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code;
    RETURN ret;
END;
```

## Name

personne\_info\_text\_set

## Synopsis

```
void personne_info_text_set(prm_token, prm_per_id, prm_inf_code, prm_valeur,  
prm_uti_id);
```

```
int4 prm_token;  
int4 prm_per_id;  
varchar prm_inf_code;  
text prm_valeur;  
int4 prm_uti_id;
```

## Source

```
DECLARE  
    new_pin_id integer;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    UPDATE personne_info_text  
        SET pit_valeur = prm_valeur, uti_id = prm_uti_id WHERE pit_debut ISNULL AND pit_fin ISNULL AND  
        pit_valeur IS DISTINCT FROM prm_valeur AND  
        pin_id = (SELECT pin_id FROM personne_info  
            WHERE per_id = prm_per_id AND inf_code = prm_inf_code);  
    IF NOT FOUND AND NOT EXISTS (SELECT 1 FROM personne_info_text WHERE pit_debut ISNULL AND pit_fin ISNULL  
        AND pin_id = (SELECT pin_id FROM personne_info  
            WHERE per_id = prm_per_id AND inf_code = prm_inf_code)) THEN  
        INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO :  
        INSERT INTO personne_info_text (pin_id, pit_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_uti_id);  
    END IF;  
END;
```

## Name

personne\_info\_varchar\_get

## Synopsis

```
varchar personne_info_varchar_get(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE
    ret varchar;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT piv_valeur INTO ret
        FROM personne_info_varchar
        INNER JOIN personne_info USING (pin_id)
        WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        ORDER BY piv_id DESC LIMIT 1;
    RETURN ret;
END;
```



## Name

personne\_info\_varchar\_get\_histo

## Synopsis

```
setof                               personne_info_varchar_histo  
personne_info_varchar_get_histo(prm_token, prm_per_id, prm_inf_code);
```

```
int4 prm_token;  
int4 prm_per_id;  
varchar prm_inf_code;
```

## Source

```
DECLARE  
  row personne_info_varchar_histo;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT piv_debut, piv_fin, piv_valeur,  
           login.utilisateur_prenon_nom(prm_token, uti_id)  
    FROM personne_info_varchar  
    INNER JOIN personne_info USING (pin_id)  
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code  
    ORDER BY piv_id  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

personne\_info\_varchar\_get\_multiple

## Synopsis

```
setof varchar personne_info_varchar_get_multiple(prm_token, prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
DECLARE  
  row RECORD;  
BEGIN  
  PERFORM login._token_assert (prm_token, FALSE, FALSE);  
  FOR row IN  
    SELECT piv_valeur  
    FROM personne_info_varchar  
    INNER JOIN personne_info USING (pin_id)  
    WHERE per_id = prm_per_id AND inf_code = prm_inf_code  
    ORDER BY piv_id  
  LOOP  
    RETURN NEXT row.piv_valeur;  
  END LOOP;  
END;
```

## Name

personne\_info\_varchar\_prepare\_multiple

## Synopsis

```
void      personne_info_varchar_prepare_multiple(prm_token,      prm_per_id,  
prm_inf_code);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

```
varchar prm_inf_code;
```

## Source

```
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    DELETE FROM personne_info_varchar WHERE pin_id = (SELECT pin_id FROM personne_info WHERE per_id = prm_p  
END;
```

## Name

personne\_info\_varchar\_set

## Synopsis

```
void    personne_info_varchar_set(prm_token,    prm_per_id,    prm_inf_code,
prm_valeur, prm_uti_id);
```

```
int4    prm_token;
int4    prm_per_id;
varchar prm_inf_code;
varchar prm_valeur;
int4    prm_uti_id;
```

## Source

```
DECLARE
    new_pin_id integer;
    historique boolean;
    multiple boolean;
    piv_id_dernier integer;
    the_pin_id integer;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT inf_historique INTO historique FROM meta.info WHERE inf_code = prm_inf_code;
    SELECT inf_multiple INTO multiple FROM meta.info WHERE inf_code = prm_inf_code;
    IF historique THEN
        -- Valeur historisée :
        SELECT pin_id INTO the_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO the_pin_id;
        END IF;
        SELECT piv_id INTO piv_id_dernier FROM personne_info_varchar WHERE pin_id = the_pin_id
        ORDER BY piv_id DESC LIMIT 1;
        -- on met à jour seulement si la valeur n'existe pas encore ou est différente
        IF prm_valeur IS DISTINCT FROM (SELECT piv_valeur FROM personne_info_varchar WHERE piv_id = piv_id_dernier) THEN
            -- On met la date de fin à la précédente valeur
            UPDATE personne_info_varchar SET piv_fin = CURRENT_TIMESTAMP WHERE piv_id = piv_id_dernier;
            -- puis on crée la nouvelle valeur
            INSERT INTO personne_info_varchar (pin_id, piv_valeur, piv_debut, uti_id) VALUES (the_pin_id, prm_valeur,
            END IF;
    ELSIF multiple THEN
        SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
        IF NOT FOUND THEN
            INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
        END IF;
        INSERT INTO personne_info_varchar (pin_id, piv_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_uti_id);
    ELSE
        -- Valeur ni historisée ni multiple : on écrase l'ancienne valeur
        UPDATE personne_info_varchar
            SET piv_valeur = prm_valeur, uti_id = prm_uti_id WHERE piv_debut ISNULL AND piv_fin ISNULL AND
            piv_valeur IS DISTINCT FROM prm_valeur AND
            pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code);
        IF NOT FOUND AND NOT EXISTS (SELECT 1 FROM personne_info_varchar WHERE piv_debut ISNULL AND piv_fin ISNULL AND
            pin_id = (SELECT pin_id FROM personne_info
                WHERE per_id = prm_per_id AND inf_code = prm_inf_code)) THEN
            -- ou on la crée si elle n'existe pas
            SELECT pin_id INTO new_pin_id FROM personne_info WHERE per_id = prm_per_id AND inf_code = prm_inf_code
            IF NOT FOUND THEN
                INSERT INTO personne_info (per_id, inf_code) VALUES (prm_per_id, prm_inf_code) RETURNING pin_id INTO new_pin_id;
            END IF;
            INSERT INTO personne_info_varchar (pin_id, piv_valeur, uti_id) VALUES (new_pin_id, prm_valeur, prm_uti_id);
        END IF;
    END IF;
END;
```

## Name

personne\_liste — Retourne la liste des personne d'un type donné.

## Synopsis

```
setof int4 personne_liste(prm_token, prm_ent_code);
```

```
int4 prm_token;
```

```
varchar prm_ent_code;
```

## Description

Retourne la liste des personne d'un type donné.

## Source

```
DECLARE
    row RECORD;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    FOR row IN
        SELECT per_id FROM personne WHERE ent_code = prm_ent_code ORDER BY per_id
    LOOP
        RETURN NEXT row.per_id;
    END LOOP;
END;
```

## Name

personne\_supprime — Supprime une personne.

## Synopsis

```
void personne_supprime(prm_token, prm_per_id);
```

```
int4 prm_token;
```

```
int4 prm_per_id;
```

## Description

Supprime une personne.

## Source

```
DECLARE
  row RECORD;
BEGIN
  PERFORM login._token_assert (prm_token, FALSE, FALSE);
  FOR row IN
    SELECT pin_id FROM personne_info WHERE per_id = prm_per_id
  LOOP
    DELETE FROM personne_info_boolean WHERE pin_id = row.pin_id;
    DELETE FROM personne_info_date WHERE pin_id = row.pin_id;
    DELETE FROM personne_info_integer WHERE pin_id = row.pin_id;
    DELETE FROM personne_info_integer2 WHERE pin_id = row.pin_id;
    DELETE FROM personne_info_lien_familial WHERE pin_id = row.pin_id;
    DELETE FROM personne_info_text WHERE pin_id = row.pin_id;
    DELETE FROM personne_info_varchar WHERE pin_id = row.pin_id;
    DELETE FROM personne_info WHERE pin_id = row.pin_id;
  END LOOP;
  DELETE FROM personne_info_lien_familial WHERE per_id_parent = prm_per_id;
  DELETE FROM personne_etablissement WHERE per_id = prm_per_id;
  DELETE FROM personne_groupe WHERE per_id = prm_per_id;
  DELETE FROM notes.note_usager WHERE per_id = prm_per_id;
  DELETE FROM events.event_personne WHERE per_id = prm_per_id;
  DELETE FROM personne WHERE per_id = prm_per_id;
END;
```

## Name

pgp\_key\_id

## Synopsis

```
text pgp_key_id();
```

```
bytea ;
```

## Source

```
pgp_key_id_w
```

## Name

`pgp_pub_decrypt`

## Synopsis

```
text pgp_pub_decrypt(, arg1, arg2);
```

```
bytea ;  
bytea arg1;  
text arg2;
```

## Source

```
pgp_pub_decrypt_text
```



**Name**

pgp\_pub\_decrypt

**Synopsis**

```
text pgp_pub_decrypt(, arg1, arg2, arg3);
```

```
bytea ;  
bytea arg1;  
text arg2;  
text arg3;
```

**Source**

```
pgp_pub_decrypt_text
```

---

## Name

`pgp_pub_decrypt`

## Synopsis

```
text pgp_pub_decrypt(, arg1);
```

```
bytea ;
```

```
bytea arg1;
```

## Source

```
pgp_pub_decrypt_text
```

## Name

pgp\_pub\_decrypt\_bytea

## Synopsis

```
bytea pgp_pub_decrypt_bytea(, arg1);
```

```
bytea ;  
bytea arg1;
```

## Source

```
pgp_pub_decrypt_bytea
```

**Name**

pgp\_pub\_decrypt\_bytea

**Synopsis**

```
bytea pgp_pub_decrypt_bytea(, arg1, arg2, arg3);
```

```
bytea ;  
bytea arg1;  
text arg2;  
text arg3;
```

**Source**

```
pgp_pub_decrypt_bytea
```

---

**Name**

pgp\_pub\_decrypt\_bytea

**Synopsis**

```
bytea pgp_pub_decrypt_bytea(, arg1, arg2);
```

```
bytea ;  
bytea arg1;  
text arg2;
```

**Source**

```
pgp_pub_decrypt_bytea
```

---

## Name

pgp\_pub\_encrypt

## Synopsis

```
bytea pgp_pub_encrypt(, arg1);
```

```
text ;  
bytea arg1;
```

## Source

```
pgp_pub_encrypt_text
```

## Name

`pgp_pub_encrypt`

## Synopsis

```
bytea pgp_pub_encrypt(, arg1, arg2);
```

```
text ;
```

```
bytea arg1;
```

```
text arg2;
```

## Source

```
pgp_pub_encrypt_text
```

## Name

`pgp_pub_encrypt_bytea`

## Synopsis

```
bytea pgp_pub_encrypt_bytea(, arg1);
```

```
bytea ;  
bytea arg1;
```

## Source

```
pgp_pub_encrypt_bytea
```



**Name**

pgp\_pub\_encrypt\_bytea

**Synopsis**

```
bytea pgp_pub_encrypt_bytea(, arg1, arg2);
```

```
bytea ;  
bytea arg1;  
text arg2;
```

**Source**

```
pgp_pub_encrypt_bytea
```

---

## Name

pgp\_sym\_decrypt

## Synopsis

```
text pgp_sym_decrypt(, arg1);
```

```
bytea ;  
text arg1;
```

## Source

```
pgp_sym_decrypt_text
```

## Name

`pgp_sym_decrypt`

## Synopsis

```
text pgp_sym_decrypt(, arg1, arg2);
```

```
bytea ;  
text arg1;  
text arg2;
```

## Source

```
pgp_sym_decrypt_text
```

**Name**

pgp\_sym\_decrypt\_bytea

**Synopsis**

```
bytea pgp_sym_decrypt_bytea(, arg1);
```

```
bytea ;  
text arg1;
```

**Source**

```
pgp_sym_decrypt_bytea
```

---

**Name**

pgp\_sym\_decrypt\_bytea

**Synopsis**

```
bytea pgp_sym_decrypt_bytea(, arg1, arg2);
```

```
bytea ;  
text arg1;  
text arg2;
```

**Source**

```
pgp_sym_decrypt_bytea
```

---

## Name

pgp\_sym\_encrypt

## Synopsis

```
bytea pgp_sym_encrypt(, arg1, arg2);
```

```
text ;  
text arg1;  
text arg2;
```

## Source

```
pgp_sym_encrypt_text
```

## Name

pgp\_sym\_encrypt

## Synopsis

```
bytea pgp_sym_encrypt(, arg1);
```

```
text ;  
text arg1;
```

## Source

```
pgp_sym_encrypt_text
```

**Name**

pgp\_sym\_encrypt\_bytea

**Synopsis**

```
bytea pgp_sym_encrypt_bytea(, arg1, arg2);
```

```
bytea ;  
text arg1;  
text arg2;
```

**Source**

```
pgp_sym_encrypt_bytea
```

---



## Name

pgp\_sym\_encrypt\_bytea

## Synopsis

```
bytea pgp_sym_encrypt_bytea(, arg1);
```

```
bytea ;  
text arg1;
```

## Source

```
pgp_sym_encrypt_bytea
```

## Name

pgprocedures\_add\_call

## Synopsis

```
void pgprocedures_add_call(prm_method, prm_nargs);
```

```
varchar prm_method;
```

```
int4 prm_nargs;
```

## Source

```
BEGIN
  UPDATE pgprocedures_stats SET cal_ncalls = cal_ncalls + 1 WHERE cal_function_name = prm_method AND cal_
  IF NOT FOUND THEN
    INSERT INTO pgprocedures_stats (cal_function_name, cal_nargs, cal_ncalls) VALUES (prm_method, prm_nar
  END IF;
END;
```

## Name

pgprocedures\_search\_arguments

## Synopsis

```
setof pgprocedures_search_arguments(prm_function);
```

*varchar prm\_function*;

## Source

```
DECLARE
row pgprocedures_search_arguments%ROWTYPE;
BEGIN
FOR row IN
SELECT proargnames, proargtypes
FROM pg_proc
WHERE proname = prm_function
ORDER BY pronargs DESC
LOOP
RETURN NEXT row;
END LOOP;
END;
```

## Name

pgprocedures\_search\_function

## Synopsis

```
pgprocedures_search_function          pgprocedures_search_function(prm_method,  
prm_nargs);
```

```
varchar prm_method;  
int4 prm_nargs;
```

## Source

```
DECLARE  
    ret pgprocedures_search_function%ROWTYPE;  
BEGIN  
    --PERFORM pgprocedures_add_call (prm_method, prm_nargs);  
    SELECT  
        pg_namespace_proc.nspname AS proc_nspname,  
        proargtypes,  
        prorettype,  
        pg_type_ret.typtype AS ret_typtype,  
        pg_type_ret.typname AS ret_typname,  
        pg_namespace_ret.nspname AS ret_nspname,  
        proretset  
    INTO ret  
    FROM pg_proc  
        INNER JOIN pg_type pg_type_ret ON pg_type_ret.oid = pg_proc.prorettype  
        INNER JOIN pg_namespace pg_namespace_ret ON pg_namespace_ret.oid = pg_type_ret.typnamespace  
        INNER JOIN pg_namespace pg_namespace_proc ON pg_namespace_proc.oid = pg_proc.pronamespace  
    WHERE proname = prm_method AND pronargs = prm_nargs;  
    RETURN ret;  
END;
```

## Name

`pour_code` — Retourne une chaîne sans caractères non-ascii.

## Synopsis

```
varchar pour_code(str);
```

```
varchar str;
```

## Description

Retourne une chaîne sans caractères non-ascii.

## Source

```

DECLARE
BEGIN
  RETURN LOWER (TRANSLATE (str, ' ÀÁÂÃÄÅàáâãäåöøÙÚÛÜÝÞßàáâãäåöøÈÉÊËèéêëççìíîïìíîùúûüýñ°«»#,()'+/.&''',
    '_aaaaaaaaaaaaooooooooooooooooeeeeeeeeeecciiiiiiiiuuuuuuuynn__d_____
END;
```

## Name

`prise_en_charge_select` — Retourne la liste des groupes de prise en charge auxquels un utilisateur a accès depuis le portail d'un établissement.

## Synopsis

```
setof prise_en_charge_select prise_en_charge_select(prm_token, prm_eta_id);
```

```
int4 prm_token;  
int4 prm_eta_id;
```

## Description

Retourne la liste des groupes de prise en charge auxquels un utilisateur a accès depuis le portail d'un établissement.

## Source

```
DECLARE  
    uti integer;  
    row prise_en_charge_select;  
BEGIN  
    PERFORM login._token_assert (prm_token, FALSE, FALSE);  
    SELECT uti_id INTO uti FROM login.token WHERE tok_token = prm_token;  
    FOR row IN SELECT groupe.grp_id, groupe.grp_nom FROM groupe  
        INNER JOIN login.grouputil_groupe USING(grp_id)  
        INNER JOIN login.utilisateur_grouputil USING(gut_id) WHERE uti_id = uti AND groupe.eta_id = prm_eta_id  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

`secteur_infos_get` — Retourne les informations supplémentaires sur un secteur.

## Synopsis

```
secteur_infos secteur_infos_get(prm_token, prm_sec_id);
```

```
int4 prm_token;
```

```
int4 prm_sec_id;
```

## Description

Retourne les informations supplémentaires sur un secteur.

## Source

```
DECLARE
    ret secteur_infos;
BEGIN
    PERFORM login._token_assert (prm_token, FALSE, FALSE);
    SELECT * INTO ret FROM secteur_infos WHERE sec_id = prm_sec_id;
    RETURN ret;
END;
```

## Name

secteur\_infos\_update — Modifie les informations supplémentaires sur un secteur.

## Synopsis

```
void secteur_infos_update(prm_token, prm_sec_id, prm_editable);
```

```
int4 prm_token;  
int4 prm_sec_id;  
bool prm_editable;
```

## Description

Modifie les informations supplémentaires sur un secteur.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  UPDATE secteur_infos SET sec_editable = prm_editable WHERE sec_id = prm_sec_id;  
  IF NOT FOUND THEN  
    INSERT INTO secteur_infos (sec_id, sec_editable) VALUES (prm_sec_id, prm_editable);  
  END IF;  
END;
```

# 15. ressource

## 15.1. Description

Ressources à affecter à des événements.

## 15.2. Tables

### 15.2.1. ressource.ressource

Les ressources.

res\_id (int4)

res\_nom (varchar)

### Liens vers cette table

- event\_ressource.res\_id
- ressource\_secteur.res\_id

### 15.2.2. ressource.ressource\_secteur

Affectation des ressources à des secteurs.

rse\_id (int4)

res\_id (int4)  
Clé étrangère vers ressource.res\_id

sec\_id (int4)  
Clé étrangère vers secteur.sec\_id



## **15.3. Types**

### **15.3.1. ressource.ressource\_liste\_details**

## **15.4. Fonctions**

## Name

ressource\_add — Ajoute une ressource.

## Synopsis

```
int4 ressource_add(prm_token, prm_res_nom);
```

```
int4 prm_token;
```

```
varchar prm_res_nom;
```

## Description

Ajoute une ressource.

## Source

```
DECLARE
  ret integer;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  INSERT INTO ressource.ressource (res_nom) VALUES (prm_res_nom) RETURNING res_id INTO ret;
  RETURN ret;
END;
```

## Name

ressource\_get — Retourne les informations d'une ressource.

## Synopsis

```
ressource ressource_get(prm_token, prm_res_id);
```

```
int4 prm_token;  
int4 prm_res_id;
```

## Description

Retourne les informations d'une ressource.

## Source

```
DECLARE  
  ret ressource.ressource;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  SELECT * INTO ret FROM ressource.ressource WHERE res_id = prm_res_id;  
  RETURN ret;  
END;
```

## Name

ressource\_list — Retourne la liste des ressources.

## Synopsis

```
setof ressource ressource_list(prm_token, prm_sec_id);
```

```
int4 prm_token;
```

```
int4[] prm_sec_id;
```

## Description

Retourne la liste des ressources.

## Source

```
DECLARE
  row ressource.ressource;
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  FOR row IN
    SELECT DISTINCT ressource.* FROM ressource.ressource
      INNER JOIN ressource.ressource_secteur USING(res_id)
      WHERE sec_id = ANY(prm_sec_id)
      ORDER BY res_nom
  LOOP
    RETURN NEXT row;
  END LOOP;
END;
```

## Name

ressource\_liste\_details — Liste en détail les ressources.

## Synopsis

```
setof      ressource_liste_details      ressource_liste_details(prm_token,  
prm_sec_id);
```

```
int4 prm_token;  
int4 prm_sec_id;
```

## Description

Liste en détail les ressources.

## Source

```
DECLARE  
  row ressource.ressource_liste_details;  
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  FOR row IN  
    SELECT res_id, res_nom, concatenate (besoins.sec_nom)  
    FROM ressource.ressource  
    LEFT JOIN ressource.ressource_secteur USING(res_id)  
    LEFT JOIN meta.secteur besoins ON besoins.sec_id = ressource_secteur.sec_id  
    WHERE (prm_sec_id ISNULL OR prm_sec_id = besoins.sec_id)  
    GROUP BY res_id, res_nom  
  LOOP  
    RETURN NEXT row;  
  END LOOP;  
END;
```

## Name

ressource\_save — Enregistre les informations d'une ressource.

## Synopsis

```
void ressource_save(prm_token, prm_res_id, prm_nom);
```

```
int4 prm_token;  
int4 prm_res_id;  
varchar prm_nom;
```

## Description

Enregistre les informations d'une ressource.

## Source

```
BEGIN  
  PERFORM login._token_assert (prm_token, TRUE, FALSE);  
  UPDATE ressource.ressource SET res_nom = prm_nom WHERE res_id = prm_res_id;  
END;
```

## Name

ressource\_secteur\_liste — Retourne la liste des secteurs auxquels est affectée une ressource.

## Synopsis

```
setof secteur ressource_secteur_liste(prm_token, prm_res_id);
```

```
int4 prm_token;  
int4 prm_res_id;
```

## Description

Retourne la liste des secteurs auxquels est affectée une ressource.

## Source

```
DECLARE  
    row meta.secteur;  
BEGIN  
    PERFORM login._token_assert (prm_token, TRUE, FALSE);  
    FOR row IN  
        SELECT secteur.* FROM meta.secteur  
            INNER JOIN ressource.ressource_secteur USING(sec_id)  
            WHERE res_id = prm_res_id  
    LOOP  
        RETURN NEXT row;  
    END LOOP;  
END;
```

## Name

ressource\_secteur\_set — Indique à quels secteurs à est affecté une ressource.

## Synopsis

```
void ressource_secteur_set(prm_token, prm_res_id, prm_secteurs);
```

```
int4 prm_token;
```

```
int4 prm_res_id;
```

```
varchar[] prm_secteurs;
```

## Description

Indique à quels secteurs à est affecté une ressource.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM ressource.ressource_secteur WHERE res_id = prm_res_id;
  IF prm_secteurs NOTNULL THEN
    FOR i IN 1 .. array_upper(prm_secteurs, 1) LOOP
      INSERT INTO ressource.ressource_secteur(res_id, sec_id) VALUES (prm_res_id, (SELECT sec_id FROM met
    END LOOP;
  END IF;
END;
```



## Name

ressource\_supprime — Supprime une ressource.

## Synopsis

```
void ressource_supprime(prm_token, prm_res_id);
```

```
int4 prm_token;
```

```
int4 prm_res_id;
```

## Description

Supprime une ressource.

## Source

```
BEGIN
  PERFORM login._token_assert (prm_token, TRUE, FALSE);
  DELETE FROM ressource.ressource_secteur WHERE res_id = prm_res_id;
  DELETE FROM ressource.ressource WHERE res_id = prm_res_id;
END;
```