

S2.04 Exploitation d'une Base de Données

Aymen Alloune

Patrick Chen

Maurice Prieto

<!-- Pour information en plus de ce fichier il y aura un autre fichier contenant tout le code de la base de données. -->

Sommaire:

I. Modélisation de données

I.1 Diagramme UML / PAGE 2

I.2 Script de création / PAGE 3

II: Visualisation de données

II.1: Définition d'un ensemble de données dérivées / PAGE 5

II.2 : Description de procédures et vues / PAGE 7

II.2.1 : Les vues / PAGE 5

II.2.2 : Les procédures / PAGE 11

III : Description de procédures et vues

III.1 : Définition des règles d'accès / PAGE 19

III.2 : Définition des règles d'accès / PAGE 20

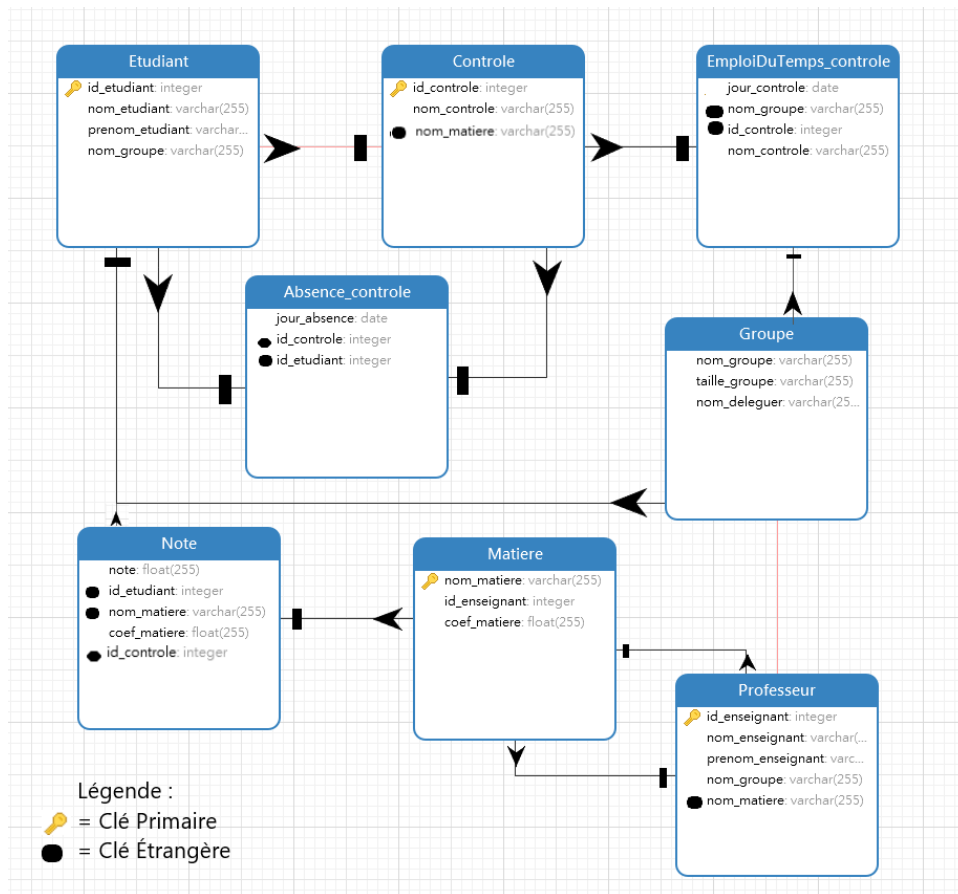
III.2.1 : Procédures d'accès pour étudiant / PAGE 20

III.2.2 : Procédures d'accès pour professeur /PAGE23

Première Partie I : La modélisation de nos données

I.1 Diagramme UML :

Juste en dessous on retrouve le diagramme UML de la base de données



I.2 Script de création :

Ensuite juste en dessous on retrouve toutes les tables de la base de données de gestion de note d'étudiant

```
CREATE TABLE Etudiant
(
  id_etudiant integer PRIMARY KEY,
  nom_etudiant varchar(10),
  prenom_etudiant varchar(10),
  nom_groupe varchar(10)
);

CREATE TABLE Matiere(
  nom_matiere varchar(10) PRIMARY KEY,
  id_enseignant integer,
  coef_matiere float
);

CREATE TABLE Professeur
(
  id_enseignant integer PRIMARY KEY,
  nom_enseignant varchar(10) ,
  prenom_enseignant varchar(10) ,
  nom_groupe varchar(10),
  nom_matiere varchar(10),
  FOREIGN KEY (nom_matiere) REFERENCES Matiere(nom_matiere)
);

CREATE TABLE Controle (
  id_controle integer PRIMARY KEY,
  nom_controle varchar(20),
  nom_matiere varchar(20), /*NEW*/
  FOREIGN KEY (nom_matiere) REFERENCES Matiere(nom_matiere)
);
```

```
CREATE TABLE Note
(
note float NOT NULL,
id_etudiant integer,
nom_matiere varchar(10) ,
coef_matiere float,
id_controle integer, /*FOREIGN NEW*/
FOREIGN KEY (id_etudiant) REFERENCES Etudiant(id_etudiant),
FOREIGN KEY (nom_matiere) REFERENCES Matiere(nom_matiere),
FOREIGN KEY (id_controle) REFERENCES Controle(id_controle)
);
```

```
CREATE TABLE Groupe
(
nom_groupe varchar(20) NOT NULL PRIMARY KEY ,
taille_groupe integer,
nom_deleguer varchar(20)
);
```

```
CREATE TABLE EDT_controle
(
jour_controle date NOT NULL,
id_controle integer,
nom_controle varchar(20),
nom_groupe varchar(20),
FOREIGN KEY (id_controle) REFERENCES Controle(id_controle),
FOREIGN KEY (nom_groupe) REFERENCES Groupe(nom_groupe)
);
```

```
CREATE TABLE Absence_controle
(
jour_absence date NOT NULL,
id_controle integer,
```

```
id_etudiant integer,  
FOREIGN KEY (id_etudiant) REFERENCES Etudiant(id_etudiant),  
FOREIGN KEY (id_controle) REFERENCES Controle(id_controle)  
);
```

Deuxième partie II: Visualisation de données

II.1: Définition d'un ensemble de données dérivées

Ici chaque donnée a été créée dans un but précis pour qu'elle nous aide à essayer chaque vue et procédure qui arrivent par la suite, chaque table reçoit des données dérivées.

```
INSERT INTO Etudiant values (5832,'Alloune','Aymen','Pegasus');  
INSERT INTO Etudiant values (5733,'Chen','Patrick','Pegasus');  
INSERT INTO Etudiant values (5266,'Prieto','Mauricio','Pegasus');  
INSERT INTO Etudiant values (5183,'Dujardin','Jean','Draco');  
INSERT INTO Etudiant values (5983,'Dupot','Jeanne','Draco');  
INSERT INTO Etudiant values (5003,'Pierrot','Payera','Andromeda');
```

```
INSERT INTO Matiere values ('BD',8533,2.5);  
INSERT INTO Matiere values ('Java',8749,4);  
INSERT INTO Matiere values ('HTML',8652,3);
```

```
INSERT INTO Professeur values (8533,'Abir','Ab.','Pegasus');  
INSERT INTO Professeur values (8749,'Azzag','Az.','Andromeda');  
INSERT INTO Professeur values (8652,'Jean','Lassale','Draco');
```

```
INSERT INTO Controle values (1,'Test_BD','BD');
```

```

INSERT INTO Controle values (2,'Test_Java','Java');
INSERT INTO Controle values (3,'Test_HTML','HTML');

INSERT INTO Note values (16.5,5832,'BD',2.5,1);
INSERT INTO Note values (17,5832,'Java',4,2);
INSERT INTO Note values (15,5832,'HTML',3,3);
INSERT INTO Note values (19.99,5733,'Java',4,2);    /* GROUPE PEGASUS */
INSERT INTO Note values (20,5266,'HTML',3,3);

INSERT INTO Note values (17,5183,'BD',2.5,1);
INSERT INTO Note values (17.75,5983,'BD',2.5,1);    /* GROUPE DRACO */

INSERT INTO Note values (15.5,5003,'HTML',3,3);    /* GROUPE ANDROMEDA */
INSERT INTO Note values (16.5,5003,'BD',2.5,1);

INSERT INTO Groupe values ('Pegasus',25,'Ony');
INSERT INTO Groupe values ('Draco',25,'Dujardin');
INSERT INTO Groupe values ('Andromeda',26,'Pierrot');

INSERT INTO EDT_controle values ('2022-04-19',1,'Test_BD', 'Pegasus');
INSERT INTO EDT_controle values ('2022-05-10',2,'Test_Java', 'Andromeda');
INSERT INTO EDT_controle values ('2022-05-26',3,'Test_HTML', 'Draco');

INSERT INTO Absence_controle values ('2022-04-19',1,5832);
INSERT INTO Absence_controle values ('2022-05-10',2,5832);
INSERT INTO Absence_controle values ('2022-05-10',2,5733);
INSERT INTO Absence_controle values ('2022-05-26',3,5266);

```

II.2 : Description de procédures et vues

En regroupant toutes les vues et procédures que nous allons présenter pour chaque table et données dérivées qu'il y aura, forcément on trouvera également un moyen d'accéder à ses données.

II.2.1 : Les vues

A/ Etudiant grouper par groupe

Cette vue nous donne accès à chaque élève ordonné par groupe

```
Create VIEW etudiant_groupe as select distinct
e.nom_groupe,id_etudiant,nom_etudiant
from Etudiant e, Groupe g
where e.nom_groupe=g.nom_groupe;
```

Draco	5183	Jean
-------	------	------

B/ Moyenne des étudiants par matière

Une vue qui regroupe les étudiants par la moyenne qu'ils ont eu dans une matière.

```
Create VIEW moy_matiere as select * from (
    select avg(note)as Note,e.id_etudiant,e.nom_etudiant,nom_matiere
from Note n , Etudiant e where n.id_etudiant=e.id_etudiant
group by nom_matiere,e.id_etudiant,e.nom_etudiant order by Note DESC
) as s order by nom_matiere DESC;
```

17	5183	Jean	BD
----	------	------	----

C/ Les moyennes des groupes

Ici on regroupe les moyenne par groupe

```
Create VIEW moy_groupe_note as select avg(note) as Note,nom_groupe
from Note n, Etudiant e
where n.id_etudiant=e.id_etudiant group by nom_groupe;
```

(3 lignes)

D/ Moyenne des déléguer

Cette vue nous montre les moyennes des déléguer

```
Create VIEW moy_deleguer as select avg(note) as  
Note,nom_deleguer,g.nom_groupe  
from Etudiant e , Note n , Groupe g  
where g.nom_deleguer=e.nom_etudiant and e.id_etudiant=n.id_etudiant  
group by nom_deleguer,g.nom_groupe order by Note DESC;
```

16	Pierrot	Andromeda
----	---------	-----------

E/ Moyenne de groupe par controle

La prochaine vue regroupe les groupes ayant eu les meilleur notes , regrouper par contrôle fait.

```
Create view moy_groupe_controle as select avg(note) as  
Note,nom_groupe,c.id_controle  
from Note n, Controle c,Etudiant e  
where n.id_controle=c.id_controle and n.id_etudiant=e.id_etudiant  
group by c.id_controle,nom_groupe order by c.id_controle;
```

```
postgres=# select * from moy_groupe_controle;  
note | nom_groupe | id_controle  
-----+-----+-----  
16.5 | Andromeda | 1  
17.375 | Draco | 1  
16.5 | Pegasus | 1  
19.99 | Pegasus | 2  
15.5 | Andromeda | 3  
20 | Pegasus | 3  
(6 lignes)
```

F/ Absence par contrôle des groupes

Pour celle-ci on va compter le nombre d'absent a chaque contrôle et les trier par groupe

```
CREATE VIEW absence_controle_groupe as SELECT count(jour_absence) as  
Jour_abs,nom_groupe
```



```
from Absence_controle a, Etudiant e
where a.id_etudiant=e.id_etudiant group by nom_groupe order by Jour_abs DESC;
```

```
postgres=# select * from absence_controle_groupe;
 jour_abs | nom_groupe
-----+-----
         2 | Pegasus
         1 | Andromeda
         1 | Draco
(3 lignes)
```

G/ Absence d'étudiant décroissant

Ici on va trier les absent a chaque contrôle et les trier par ordre décroissant et voir qui sont les élèves les plus absent.

```
CREATE VIEW abs_etudiant_desc as SELECT count(jour_absence), nom_etudiant
from Absence_controle a , Etudiant e
where a.id_etudiant=e.id_etudiant group by nom_etudiant order by nom_etudiant
DESC;
```

```
postgres=# select * from abs_etudiant_desc;
 count | nom_etudiant
-----+-----
      1 | Pierrot
      1 | Dujardin
      1 | Chen
      1 | Alloune
(4 lignes)
```

H/ Moyenne décroissante

Les moyennes de tous les étudiant trier dans l'ordre décroissant mettant en valeur les meilleur élèves.

```
CREATE VIEW moy_etudiant as SELECT avg(note) as Note, nom_etudiant
```

```
from Etudiant e , Note n
where e.id_etudiant=n.id_etudiant group by nom_etudiant order by Note DESC;
```

```
postgres=# select * from moy_etudiant;
note | nom_etudiant
-----+-----
    20 | Prieto
  19.99 | Chen
  17.75 | Dupot
    17 | Dujardin
   16.5 | Alloune
    16 | Pierrot
(6 lignes)
```

I/ Les meilleurs professeurs

Enfin pour cette dernière vue on s'attaque à la table des professeurs, on va les trier selon les notes qu'on eut leurs élèves à chaque contrôle.

```
CREATE VIEW Prof_Perf as SELECT nom_enseignant, avg(note) as Note_etudiant
from Etudiant e, Professeur p , Note n
where p.nom_groupe=e.nom_groupe and e.id_etudiant=n.id_etudiant
group by nom_enseignant order by Note_etudiant DESC;
```

```
postgres=# select * from prof_perf;
nom_enseignant | note_etudiant
-----+-----
Abir           |          18.83
Azzag          |          18.83
Jean           |          17.375
(3 lignes)
```

II.2.2 : Les procédures

A/ *Première procédure*, cette procédure appeler Note_CE qui signifie Note coefficients étudiants va nous permettre d'effectuer le calcul de coefficients avec d'une part les notes simples et de l'autre cotée les coefficients les deux donner en paramètre de la fonction, au final cela nous retournera la moyenne calculer grâce à la formule de calcul de coefficient.

```
CREATE or REPLACE function Note_CE (in note float[], in coef float[])
RETURNS float
as
$$
DECLARE
```

```

note_calculer float;
n int;
som_notecoeff float;
somme_coef float;
BEGIN
som_notecoeff:=0;
somme_coef:=0;
IF(array_upper($1,1)!=array_upper($2,1) ) THEN
note_calculer:=0;
raise notice'<!> La taille des tableaux est pas pareil <!>';
return note_calculer;
END IF;
/* Remplissage de la variable som_notecoeff, en faisant la somme de chaque note
multiplier par leur coefficients respectifs */
FOR n in 1 .. array_upper($1,1) LOOP
som_notecoeff:=som_notecoeff+($1[n]*$2[n]);
END LOOP;
/* Remplissage de la variable somme_coef , ici on fait juste la somme de tout
les coefs */
FOR n in 1 .. array_upper($2,1) LOOP
somme_coef:=somme_coef+coef[n];
END LOOP;
/* enfin le calcul final on divise som_notecoeff par la somme des coefficients,
somme_coef*/
note_calculer:=som_notecoeff/somme_coef;
return note_calculer;
END;
$$ language plpgsql;

```

Voici le test :

```
postgres=# select * from note_ce(ARRAY[15,12,16],ARRAY[2,1,3]);
note_ce
-----
      15
(1 ligne)
```

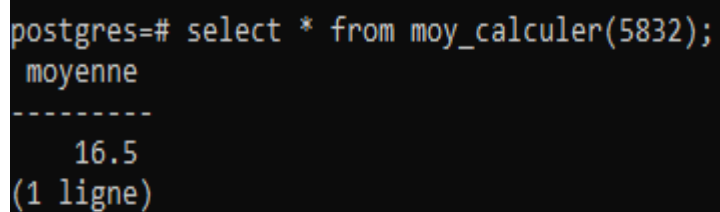
B/ *Seconde procédures* , procédures appeler Moyenne calculer va faire appel a la procédure d'au dessus, elle va prendre en parametre simplement l'id d'un etudiant puis va créer deux tableaux de float, et va les remplir des notes de l'étudiant dans un premier tableaux puis des coefficients de ces même notes dans un second tableau, maintenant avec ces deux tableaux elle va appeler la fonction précédente faire le calcul et retourner le résultat.

```
CREATE OR REPLACE function Moy_calculer(in id int,out moyenne float)
returns float
as
$$
DECLARE
/* Déclaration d'un curseur totalement lié pour une requête particulière */
curseur CURSOR FOR
select note,coef_matiere from Note e, etudiant e2 where
e.id_etudiant=e2.id_etudiant and e.id_etudiant=$1 order by note DESC;
note_array float[];
note_coef float[];
tmp_note float;
tmp_coef float;
i int;
test_etudiant int;
BEGIN
with test_etudiant As(
    select id_etudiant from etudiant where id_etudiant=$1
)
select count(*) into test_etudiant from test_etudiant;
IF (test_etudiant=0) THEN
raise notice '<!'> Attention id_etudiant ne correspond a personne <!'>
```

<!-- Une erreur se produit , modifier id_etudiant -->

```
';  
END IF;  
/* initialisation du compteur i */  
i:=1;  
/* On ouvre le curseur et on l'utilise pour remplir les tableaux  
note_array(représente les note simple) et note_coef */  
OPEN curseur;  
LOOP  
FETCH curseur INTO tmp_note,tmp_coef;  
EXIT WHEN NOT FOUND;  
note_array:=array_append(note_array,tmp_note);  
note_coef:=array_append(note_coef,tmp_coef);  
i:=i+1;  
END LOOP;  
CLOSE curseur;  
select * into moyenne from Note_CE(note_array,note_coef);  
END;  
$$ language plpgsql;
```

Voici le test:



```
postgres=# select * from moy_calculer(5832);  
moyenne  
-----  
      16.5  
(1 ligne)
```

C/ troisième procédure, Pour cette troisième procédure on va pouvoir avoir un œil sur les notes d'un étudiant avec un petit récapitulatif regroupant le minimum et maximum ainsi que le nombre de notes qu'il a et sa moyenne.

```
CREATE or REPLACE function apercu_note

(int,out Nombre_note float,out Note_min float, out Moy_note float, out
Note_max float)

returns setof record

as

$$

With note_max As

(select note as max from note where id_etudiant = $1 order by max DESC limit
1)

,note_min as

(select note as min from note where id_etudiant = $1 order by min ASC limit 1)

,note_moy as

(select avg(note) as moy from note where id_etudiant=$1)

,nombre_note as

(select count(note) as nbre from Note where id_etudiant=$1)

SELECT nbre,min,moy,max from nombre_note,note_min,note_moy,note_max;

$$language sql;
```

Voici le test :

```
postgres=# select * from apercu_note(5832);
 nombre_note | note_min |      moy_note      | note_max
-----+-----+-----+-----
          3 |      15 | 16.166666666666668 |      17
(1 ligne)
```

D/ *quatrième procédure*, de se coter-ci on va s'occuper de la tale groupe avec laquelle on va compare un nom de groupe donner en paramètre et retourner si ce groupe et le meilleur ou non ainsi que sa moyenne de groupe et la note du meilleur groupe.

Elle nous permet également d'avoir accès au note d'un étudiant.

```
CREATE OR REPLACE function meilleur_groupe
(in varchar , out Resultat boolean,out Note_Groupe float,out
Note_MeilleurGroupe float)
returns setof record
as
$$
DECLARE
classement varchar;
note_m float;
BEGIN

/* On utilise deux CTE pour les utiliser juste apres, une pour la variable de
test , une pour la note*/

WITH meilleur_groupe as
(
select avg(note) as note,e.nom_groupe
from Note n,Etudiant e,Groupe g
where n.id_etudiant=e.id_etudiant and e.nom_groupe=g.nom_groupe
group by e.nom_groupe order by e.nom_groupe DESC limit 1
)
,noteGroupe as (select avg(note) as note
from Note n,Etudiant e,Groupe g
where n.id_etudiant=e.id_etudiant and e.nom_groupe=g.nom_groupe and
g.nom_groupe=$1
```

```

order by note DESC limit 1)

SELECT nom_groupe,m.note,n.note into
classement,Note_MeilleurGroupe,Note_Groupe from meilleur_groupe m ,noteGroupe
n;

/* Ici le test pour la variable OUT Resultat */
IF classement=$1 THEN Resultat:=true;
        ELSE Resultat:=False;

END IF;

return next;

END;

$$language plpgsql;

Voici le test :

```

```

(1 ligne)

```

E/ *cinquième procédure*, dans cette procédure on va pouvoir regrouper toutes les informations d'un contrôle précis donner grâce à l'id d'étudiant donner en paramètre.

```

CREATE or REPLACE function ControleInfo(in id int,out note float,out
Nom_Matiere varchar,out nom_controle varchar,out Jour date,out nb_Abscent int)
returns setof record

as

$$

```

```

select avg(note) as MoyenneNote,n.nom_matiere as Nom_Matiere,e.nom_controle as
NomControle,e.jour_controle as Jour,count(a.id_etudiant) as nb_Abscent

from controle c,Note n,EDT_controle e,Absence_controle a

where c.id_controle=$1 and c.id_controle=n.id_controle and e.id_controle=$1
and a.id_controle=$1

group by n.Nom_Matiere,NomControle,Jour order by MoyenneNote DESC;

```

```

$$ language sql;

```

Voici le test :

F/ *sixième procédure* , pour cette avant-dernière procédure on va se concentrer sur les professeur en donnant en paramètre le nom d'un matière on obtient le nom du professeur, le groupe que le professeur prend en charge ainsi que la moyenne de ce groupe dans sa matière.

```
CREATE OR REPLACE function ProfesseurSearch(in NomMatiere varchar,out NomProf
varchar,out MoyEleves float,out GroupeEleves varchar)
returns setof record

as

$$

DECLARE

test_presence int;

BEGIN

/* Ici test pour clarifier les erreurs humaines */

with ExisteOuPas as (

    select count(nom_matiere) from matiere where nom_matiere=$1

)

select * into test_presence from ExisteOuPas;

IF (test_presence=0) THEN

    raise notice '<!> Le nom de la matiere existe pas <!>';

    return;

END IF;

/* On fait un select dans chaque variable OUT */

select nom_enseignant,avg(note),p.nom_groupe

into NomProf,MoyEleves,GroupeEleves

from Professeur p , Matiere m , Etudiant e, Note n
```

```

where m.nom_matiere=p.nom_matiere and m.nom_matiere=$1 and
e.id_etudiant=n.id_etudiant and p.nom_groupe=e.nom_groupe

group by nom_enseignant,p.nom_groupe ;

/* Return next car c'est un setof record et qu'il faut retourner qu'une seule
ligne on a donc pas besoin de boucle */

return next;

END;

$$language plpgsql;

```

Voici le test:

```
(1 ligne)
```

G/ *sixième procédure*, voici la dernière procédure , ici la spécificité est l'utilisation d'un curseur partiellement lié avec un argument celui donner en paramètre, on va avoir a la place d'un setof record , le nom de chaque étudiant renvoyer sous forme de notice et ne seulement retourner le nombre d'itération.

```

CREATE OR REPLACE function curseur_controle(in nom_controle varchar,out
NombrePrésent int)

returns int

as

$$

DECLARE

/* Utilisation d'un curseur partiellement lié */

curseur_ctl CURSOR (nom_ctl varchar) IS

select id_etudiant,nom_etudiant,e.nom_groupe

from Etudiant e , Professeur p , Controle C , Matiere m

where e.nom_groupe=p.nom_groupe and m.id_enseignant=p.id_enseignant and
m.nom_matiere=c.nom_matiere and c.nom_controle=nom_ctl

order by e.nom_groupe;

IdEtudiant int;

NomEtudiant varchar;

i int;

BEGIN

i:=0;

OPEN curseur_ctl($1);

/* Première affichage pour rendre les notices compréhensible */

```

```
raise notice E'%\t%' <-- Dans ce controle , voici la liste des eleves present
: ', $1, ' ' ;
```

```
LOOP
```

```
fetch curseur_ctl into IdEtudiant,NomEtudiant;
```

```
EXIT WHEN NOT FOUND;
```

```
/* Affichage de chaque élèves étant présent au controle donner en paramètre */
```

```
raise notice E'%\t%',IdEtudiant,NomEtudiant;
```

```
i:=i+1;
```

```
END LOOP;
```

```
CLOSE curseur_ctl;
```

```
NombrePrésent:=i;
```

```
END;
```

```
$$language plpgsql;
```

Voici le test :

```
(1 ligne)
```

III : Restrictions d'accès aux données

III.1 : Définition des règles d'accès

En premier temps de cette troisième partie on va définir des règles d'accès :

Un étudiant ne peut accéder qu'a :

- Qu'a ses informations personnelles
- Qu'a ses absences,
- Qu'a son groupe,
- Qu'a ses notes,
- Qu'a ses matières,
- Qu'a ses noms d'enseignants et non pas à leur ID
- Qu'a ses contrôles qu'il a fait
- Qu'a son EDT ;

Un professeur ne peut accéder qu'à

- A accès a uniquement ses élèves
- A accès a uniquement sa matière
- A accès a uniquement ses informations de prof
- A accès à ses contrôles dans sa matière
- A accès uniquement au absence de ses contrôles

- A accès uniquement à son emploi du temps
- Au droit d'ajouter des notes

III.2 : Procédures d'accès

III.2.1 : Procédures d'accès d'étudiant

Voici ici les procédures permettant de créer ces règles

Pour les étudiants :

```
CREATE OR REPLACE FUNCTION MesInfos(out id_etudiant int,out nom_etudiant
varchar,out prenom_etudiant varchar,out nom_groupe varchar)
returns setof record
as
$$
select id_etudiant,nom_etudiant,prenom_etudiant,nom_groupe
from Etudiant e
where e.nom_etudiant=session_user;
$$ language SQL
SECURITY DEFINER;
```

```

CREATE OR REPLACE FUNCTION MesAbsences(out jour_abs date,out id_controle int)
returns setof record
as
$$
select jour_absence,id_controle
from Absence_controle a , Etudiant e
where a.id_etudiant=e.id_etudiant and e.nom_etudiant=session_user;
$$ language SQL
SECURITY DEFINER;

```

```

CREATE OR REPLACE FUNCTION MonGroupe(out nom_groupe varchar,out taille_groupe
int,out nom_deleguer varchar)
returns setof record
as
$$
Select g.nom_groupe,taille_groupe,nom_deleguer
from Groupe g , etudiant e
where g.nom_groupe=e.nom_groupe and e.nom_etudiant=session_user;
$$ language sql
SECURITY DEFINER;

```

```

CREATE OR REPLACE FUNCTION MesNotes(out note float,out nom_matiere varchar,out
nom_controle varchar,out coef_matiere float,out id_controle int)
returns setof record
as
$$
select n.note,n.nom_matiere,c.nom_controle,n.coef_matiere,c.id_controle
from Note n, Etudiant e, Controle c

```

```
where n.id_etudiant=e.id_etudiant and e.nom_etudiant=session_user;
```

```
$$ language sql
```

```
SECURITY DEFINER;
```

```
CREATE OR REPLACE FUNCTION MesMatiere(out nom_matiere varchar,out  
nom_enseignant varchar,out coef_matiere float)
```

```
returns setof record
```

```
as
```

```
$$
```

```
select m.nom_matiere,p.nom_enseignant,m.coef_matiere
```

```
from matiere m, Etudiant e,Professeur p
```

```
where m.id_enseignant=p.id_enseignant and p.nom_groupe=e.nom_groupe and  
e.nom_etudiant=session_user;
```

```
$$ language sql
```

```
SECURITY DEFINER;
```

```
CREATE OR REPLACE FUNCTION MonEDT(out JourControle date,out id_controle  
int,out nom_controle varchar,out jour_absence date)
```

```
returns setof record
```

```
as
```

```
$$
```

```
SELECT jour_controle,edt.id_controle,nom_controle,jour_absence
```

```
from EDT_controle edt ,Absence_controle a, Etudiant e,Groupe g
```

```
where edt.nom_groupe=g.nom_groupe and g.nom_groupe=e.nom_groupe and  
e.nom_etudiant=session_user;
```

```
$$ language sql
```

```
SECURITY DEFINER;
```

III.2.2 : Procédures d'accès pour professeur

Et de ce côté les procédures pour professeur :

```
CREATE OR REPLACE FUNCTION MesInfos_Prof(out id_enseignant int,out
nom_enseignant varchar,out prenom_enseignant varchar,out nom_groupe
varchar,out nom_matiere varchar)
returns setof record
as
$$
select id_enseignant,nom_enseignant,prenom_enseignant,nom_groupe,nom_matiere
from professeur p
where p.nom_enseignant=session_user;
$$ language SQL
SECURITY DEFINER;
```

```
CREATE OR REPLACE FUNCTION MesEleves(out id_etudiant int,out nom_etudiant
varchar,out prenom_etudiant varchar,out nom_groupe varchar)
returns setof record
as
```

\$\$

```
select id_etudiant,nom_etudiant,prenom_etudiant,e.nom_groupe
from Etudiant e, Professeur p
where e.nom_groupe=p.nom_groupe and p.nom_enseignant=session_user;
$$language sql
SECURITY DEFINER;
```

```
CREATE OR REPLACE FUNCTION MesMatiere_Prof(out MesMatiere varchar,out
MesControles varchar,out coef_matiere float)
```

```
returns setof record
```

```
as
```

\$\$

```
select m.nom_matiere,c.nom_controle,m.coef_matiere
from Matiere m, Controle C, Professeur p
where m.id_enseignant=p.id_enseignant and p.nom_enseignant=session_user;
$$language sql
SECURITY DEFINER;
```

```
CREATE OR REPLACE FUNCTION MonEDT_Prof(out JourControle date,out id_controle
int,out nom_controle varchar,out NombreAbs int)
```

```
returns setof record
```

```
as
```

\$\$

```
select jour_controle,edt.id_controle,edt.nom_controle,count(jour_absence) as
NombreAbs
from EDT_controle edt,Absence_controle a,Etudiant e ,Professeur p
where edt.nom_groupe=p.nom_groupe and a.id_etudiant=e.id_etudiant and
e.nom_groupe=p.nom_groupe and p.nom_enseignant=session_user
group by jour_controle,edt.id_controle,edt.nom_controle order by NombreAbs
Desc;
$$language sql
SECURITY DEFINER;
```