Exploitation d'une Base de Données (S204)

Professeur référent : Davide Buscaldi

Groupe : Zhang Claude, Tlemsani Sofiane, Haude Aucéane et Lakrafli Ismail

Partie I - II. Modélisation de Données

Cahier des charges :

CAHIER DES CHARGES – Création d'une base de données de gestion des notes des étudiants en BUT

Dans ce projet nous mettrons en place un système de gestion des notes d'étudiants autant dans l'accessibilité que dans l'affichage et la mise à jour de celle-ci.

Objectifs :

- Mise en place d'une base de données en adéquation à la configuration de l'IUT Villetaneuse Paris 13.
- Gérer l'activité de la base de données et pouvoir y mettre des restrictions.
- Permettre la visualisation des informations pour en faire l'analyse.
- Fournir un travail technique qui résous la demande (conception, implémentation, administration, exploitation).

Le Projet de BDD S204 reste sur un périmètre universitaire car il profitera qu'aux élèves, enseignant, responsable de matière, etc... De plus dans la gestion des données celle-ci sera livré à nous et à l'enseignant qui en rajoute, C'est donné seront accessible seulement pour certaines personnes (restrictions).

Les logiciels/technologies utilisé pour ce projet seront : SQL, PLpgSQL.

Pour mieux décrire les besoins et le projet nous allons présenter le modèle de donnée et le script de la base de données pour mieux comprendre la finalité du projet.

Modèle logique de données :

Département (CodeD, nom_D)

- Cette donnée nous permet de savoir sur quel département nous sommes et enfin nous donne la voie sur le type de matière, de filières qui la constitue.

Etudiant (Etudiant_Id, Nom_E, Prénom_E, CodeD)

- Associer au département il nous permet de connaître les informations essentielles de l'élève.

Module (Module_Id, Libelle_M, Coefficient_M, Code_UE)

- Permet de référencer le module pour ensuite accéder à l'UE associé et sont type.

Filières (Filière_Id, Nom_F, CodeD)

- A partir du Département nous pouvons obtenir les filières de celui-ci.

Matière (Matiere_Id, Nom_M, Fillière_Id)

- Donne les matières disponibles par Filière sélectionné.

Enseignant (Num_Ens, Nom_Ens, Prenom_Ens)

- Nous donne les enseignants disponibles dans la base de données.

Note (Etudiant_Id, Module_Id, Num_Semestre, TD, TP, Contrôle, Rattrapge, exam,Coefficient)

- A partir de l'étudiant, du module et du semestre sélectionnées on donne tous les informations correspondantes aux notes de cette élève dans le temps impartie du semestre.

Semestre (Num_Semestre, Semestre)

- Nous donne les semestres de l'année.

UE (Code_UE, Nom_UE, coefficient, Code_TUE)

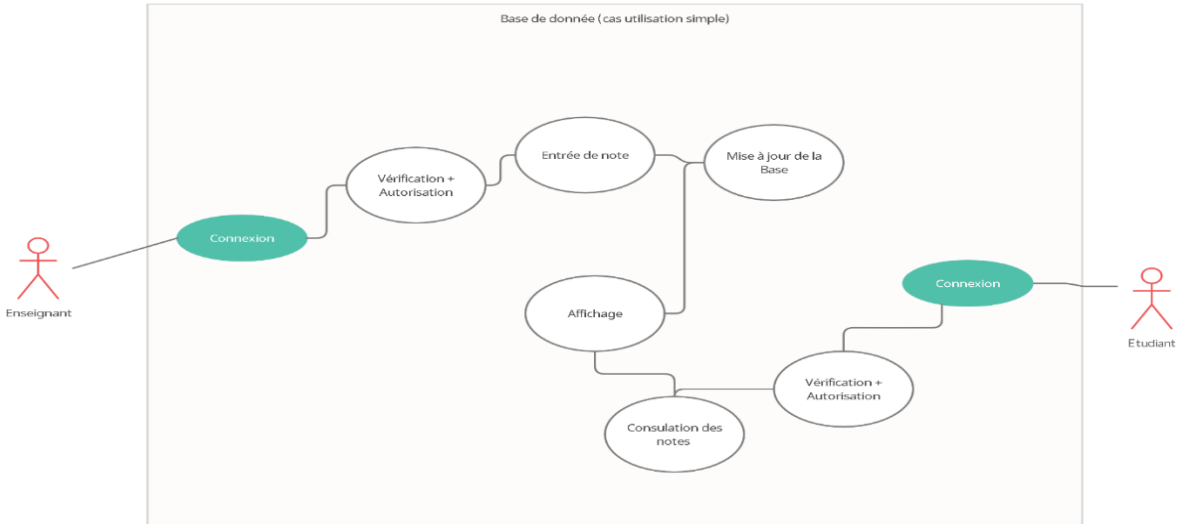
- Retourne les informations sur un Type d'UE selectionné.

Type_UE (Code_TUE, intitulé)

- Nous donne le type d'UE et son intitulé.

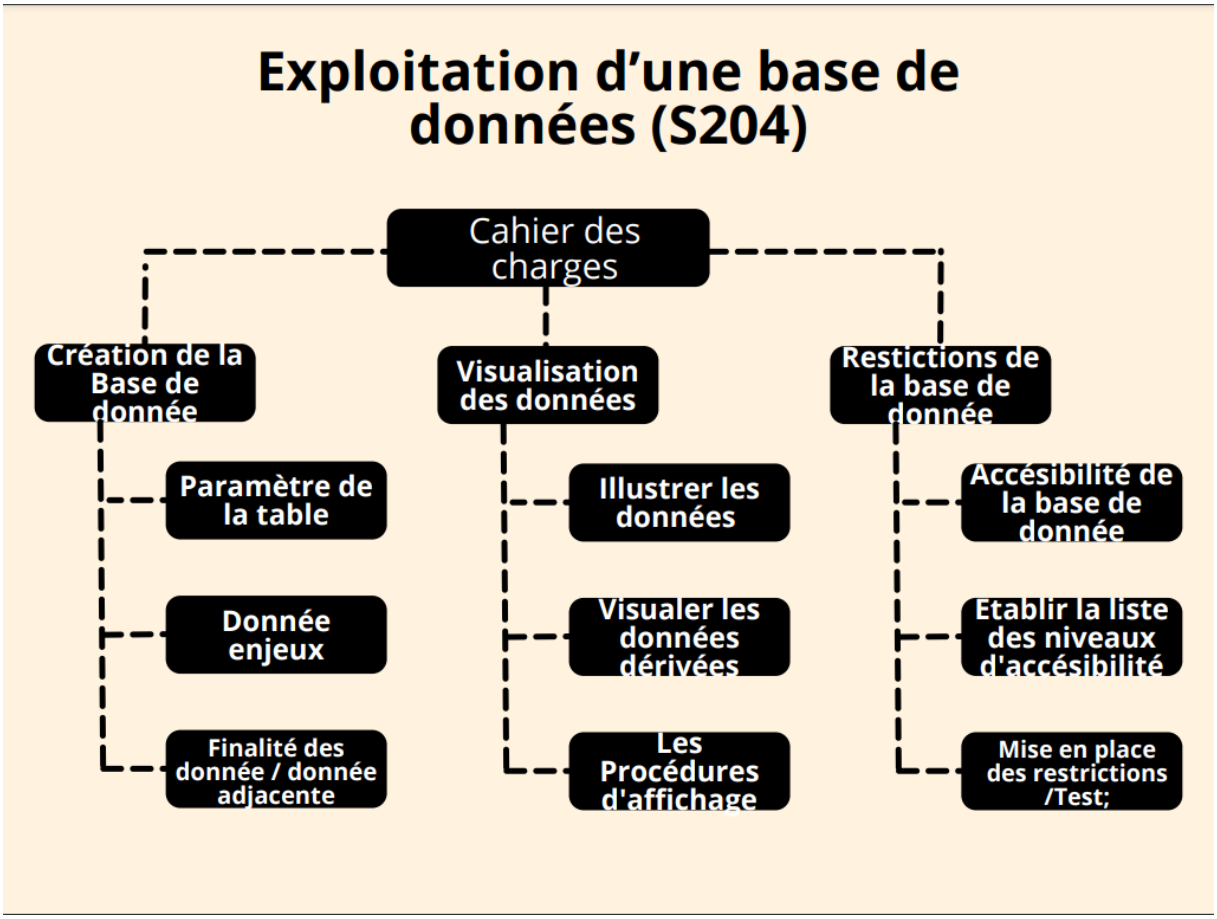
On peut constater que dans cette table beaucoup d'informations sont là pour la gestion de l'affichage des notes (Type UE, Semestre, Fillières, Module) ce qui est d'autant plus intéressant car c'est donné permettent de mieux répertorier la base de données et de la rendre plus accessible. Nous avons besoin désormais de structuré ces données et d'en faire un descriptif en analysant le script et ce qui nous permettra de construire les restrictions de cette base de données.

Nous devrons fournir cette utilité ci-dessous lors de notre production de la base de données.



Ce schéma illustre un cas d'utilisation de la base donnée ou l'enseignant lui seul peut entrer les notes qui seront ensuite afficher à l'élève.

Organigramme des tâches



Procédures stockées

Voici quelque exemple de procédure permettant de définir les règles de gestion de ces données et leurs mises en œuvre :

```
CREATE FUNCTION afficher_note(numero int)
Returns setof record as
$$
    IF numero IN Etudiant_Id from Etudiant Then
    SELECT Nom_E, TD, TP, Controle,Rattrapage,Exam from Note
```

```
        WHERE NOTE.Etudiant_Id = numero;
        ELSE IF numero IN Num_Enseignant from Enseignant
        SELECT Nom_E, TD, TP, Controle,Rattrapage,Exam from Note;
        ELSE
                PRINT ‘ Vous ne pouvez pas accéder aux notes ‘ ;
$$ language SQL ;

CREATE FUNCTION ajout_note(N_E int, note int, variable varchar(20))
Returns void as
$$
        IF N_E IN Num_Ens from Enseignant AND note between 0 AND 20
        INSERT INTO Note (variable) VALUES (note)
        ELSE
                Print ‘ Le numéro d’enseignant ou la note sont erroné ’ ;
$$ language SQL ;

CREATE FUNCTION modif_coef_UE(Code int, prof int, coef int)
Returns void as
$$
        IF prof IN Num_Enseignant from Enseignant
                UPDATE UE SET Coefficient = coef
                WHERE EXISTS (Code_UE.UE = Code) ;
$$ language SQL;

CREATE FUNCTION ajout_semestre(Numero int, S_Id int, Sem int)
Returns void as
```

```
$$
        IF Numero in Num_Ens from Enseignant AND Sem >= 0
        INSERT INTO Semestre (Semestre_Cb) VALUES (Sem)
        WHERE EXISTS (Num_Semestre.Semestre = S_Id);
$$ language SQL ;
```

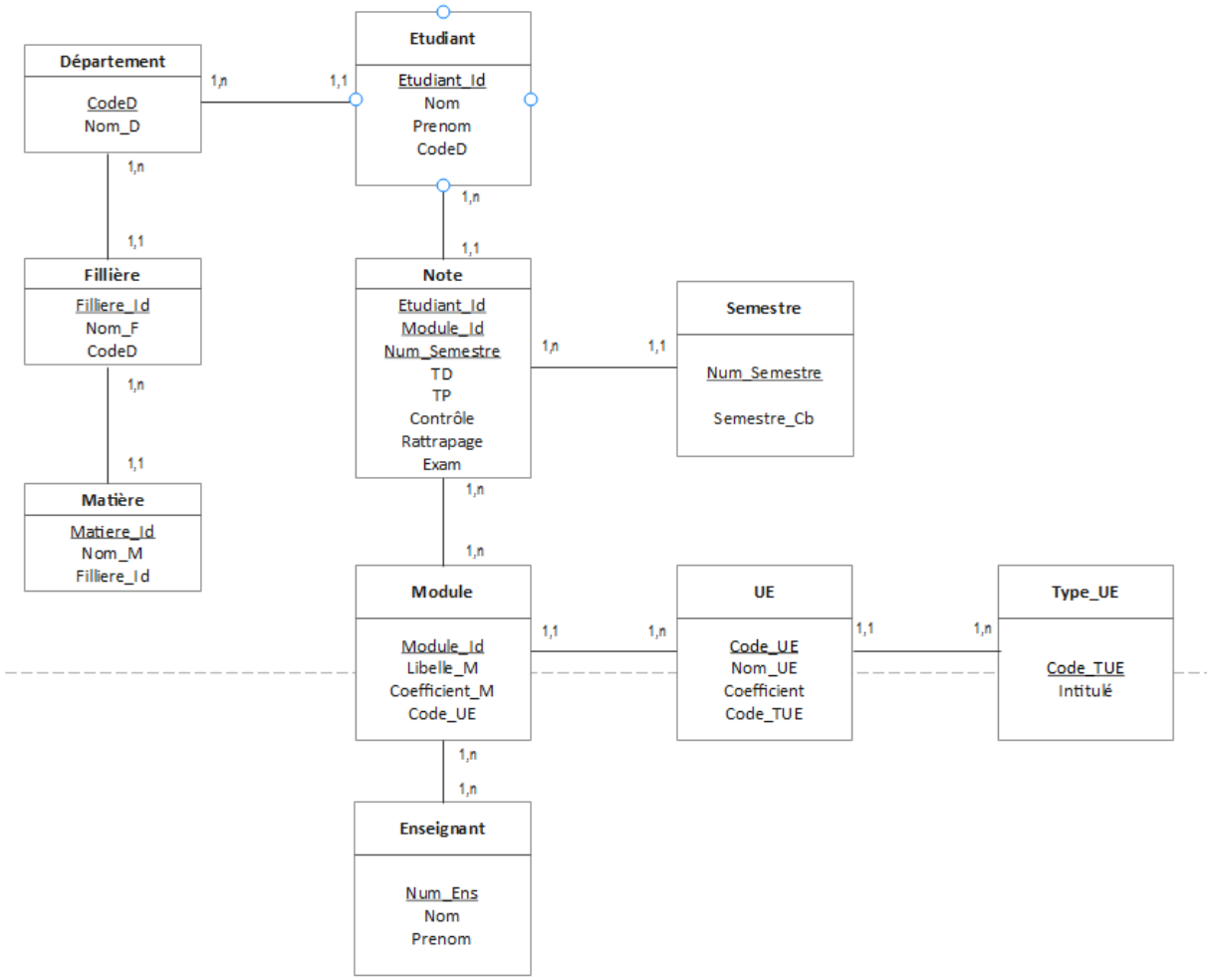
La procédure afficher_note correspond au fait qu'un élève ne peut regarder que ses propres notes, cependant pour l'enseignant lui pourra avoir accès aux notes de tous les étudiants.

La procédure ajout_note correspond au fait qu'un enseignant souhaite y insérer une nouvelle note à un étudiant, évidement les étudiants n'ont pas la permission de changer ou d'ajouter des notes.

La procédure ajout_semestre est similaire à ajout_note sauf que celle-ci correspond aux semestres (ex : S1, S2). Il est également à noter qu'un semestre ne peut être inférieur ou égal à 0 et que les notes doivent être compris entre 0 et 20.

Et enfin la procédure modif_coef_UE correspond à la modification que l'enseignant apportera au coefficient dans le cadre ou il se serait trompé sur le coefficient par exemple.

Modèle de données



Script de création du modèle de données

```
CREATE TABLE Departement
(
        CodeD serial primary key,
        Nom_D varchar (20)
) ;
```

```
CREATE TABLE Etudiant
(
        Etudiant_Id serial primary key,
        Nom_E varchar (20),
        Prenom_E varchar (20)
        CodeD int references Departement(CodeD)
);

CREATE TABLE Module
(
        Module_Id serial primary key,
        Libelle_M varchar (20),
        Coefficient_M int,
        Code_UE int references UE(Code_UE)
);

CREATE TABLE Filliere
(
        Filliere_Id serial primary key,
        Nom_F varchar (20),
        CodeD int references Departement(CodeD)
) ;

CREATE TABLE Matiere
(
        Matiere_Id serial primary key,
```

```
	Nom_M varchar(20),
	Filliere_Id int references Filliere(Filliere_id)
);

CREATE TABLE Enseignant
(
	Num_Ens serial primary key,
	Nom_Ens varchar(20),
	Prenom_Ens varchar(20)
);

CREATE TABLE Note
(
	Etudiant_Id int references Etudiant (Etudiant_Id),
	Module_Id int references Module (Module_Id),
	Num_Semestre int references Semestre(Num_Semestre),
	TD int,
	TP int,
	Controle int,
	Rattrapage int,
	Exam int,
	Coefficient int
);

CREATE TABLE Semestre
(
```

```
        Num_Semestre serial primary key,
        Semestre_Cb int
);
CREATE TABLE UE
(
        Code_UE serial primary key,
        Nom_UE varchar(20),
        Coefficient int,
        Code_TUE int references Type_UE(Code_TUE)
);

CREATE TABLE Type_UE
(
        Code_TUE serial primary key,
        Intitule varchar(20)
) ;
```

Partie II - III. Visualisation de Données

La vue Moyennes_matiere permet d'avoir une vue des moyennes des élèves par matière.

```
CREATE VIEW Moyennes_matiere
 AS
        SELECT e.Etudiant_id , Nom_E, Prenom_E, m.Matiere_id,
                avg(note) as moyenne
        FROM Etudiant e, Matiere m, Controle c, Notes n
         WHERE m.Matiere_id=c.Matiere id
          AND c.Controle_id =n.Controle_id

 AND n. Etudiant id =e. Etudiant id
 GROUP BY e.etudiant_id, Nom, Prenom, m.Matiere id
```

Les vues module_desc et asc permettent d'avoir une vue des notes des modules par ordres décroissant et croissant.

```
CREATE VIEW module_desc

AS

SELECT e.Etudiant_id, Nom,Prenom,m.Module_id, n.Module_id as module_desc

FROM Etudiant e , Module m,Note n

ORDER BY n.Module_id DESC ;


CREATE VIEW module_asc

AS

SELECT e.Etudiant_id, Nom,Prenom,m.Module_id, n.Module_id as module_desc

FROM Etudiant e , Module m,Note n

ORDER BY n.Module_id ASC ;
```

La procedure fil_etud permet d'afficher l'ensemble des noms des étudiants étant dans un département.

```
CREATE or REPLACE FUNCTION fil_etud( inout Fillière varchar,out tnom varchar[])
DECLARE
nom Etudiant.Nom_E%TYPE;
fil Fillière.Nom_F%TYPE;
Filliere_cur CURSOR (f varchar) FOR
SELECT Nom_F,Nom_E
FROM Etudiant,Fillière
WHERE Nom_F = f;
BEGIN
        OPEN Filliere_cur (Fillière);
        tnom:='{}'::varchar;
        LOOP
                FETCH Filliere_cur INTO fil,nom;
                EXIT WHEN NOT FOUND;
                tnom:= array_append(tnom,nom);
        END LOOP;
        CLOSE Filliere_cur;
END;
$$ LANGUAGE plpgsql;
```

La procédure grade_exam permet d'afficher un ensemble d'élève ayant eu une note supérieure ou égale à 10 aux examens.

```
CREATE or REPLACE FUNCTION grade_exam( inout exam varchar, out tnom varchar[ ] )
RETURNS record
```

```
AS $$
DECLARE
        Nom Etudiant.Prenom_E%TYPE ;
        Note Note.Exam%TYPE ;
        grade_cur CURSOR (g varchar) FOR SELECT Prenom_E, Exam from Note natural
join Etudiant WHERE Exam >= 10 ;

BEGIN
        OPEN grade_cur(grade) ;
        tnom : = '{}' :: varchar ;
        LOOP
                FETCH grade_cur INTO NOTE, NOM ;
                EXIT WHEN NOT FOUND ;
                tnom : = array_append(tnom,nom) ;
        END LOOP;
        CLOSE grade_cur ;
END ;
$$ language plpgsql ;
```

La procédure noteEtudP permet de retourner la note d'un étudiant pris en paramètre

```
CREATE or REPLACE FUNCTION noteEtudP (varchar)
        Returns decimal(4,2)
as $$
        DECLARE
                note decimal(4,2) ;
        BEGIN
                Select TD, TP, Contrôle, Rattrapage, Exam from Note natural join Etudiant
```

```
                    WHERE Prenom_E = $1 ;
            Return note ;
END ;
$$ language plpgsql ;
```

La procedure my_data permet à chaque étudiant de visualiser les données qui lui sont associés (nom, prénom, note …).

```
CREATE or REPLACE FUNCTION my_data( out prenom_e varchar, out libelle_m varchar , out examen numeric, out control numeric)
AS $$
BEGIN
    SELECT Prenom_E, Libelle_M, Exam, Controle into prenom_e, libelle_m, examen, control FROM Etudiant, Note, Module
    WHERE session_user = lower(enom) :: name ;
END ;
$$ language PLpgSQL
```

Partie III - IV. Restrictions d'accès aux Données

La procédure entre_note permet d'autoriser seulement les enseignant à insérer ou supprimer des notes.

```
CREATE or replace FUNCTION entre_note()

RETURNS TRIGGER AS

$$

        BEGIN
                IF current_user  in Enseignant.Nom_Ens  THEN
                        IF TG_OP='INSERT' THEN
                                return NEW;
                        END IF;
                        IF TG_OP='DELETE' THEN
                return OLD;
$$ LANGUAGE plpgsql;

CREATE TRIGGER entre_note
        BEFORE
        INSERT or DELETE on Note
        for EACH ROW
        EXECUTE PROCEDURE entre_note ();
```

La procedure MesResultats permet à son utilisateur (étudiant) d'avoir accès seulement à ces notes. Si l'utilisateur n'est pas un étudiant cela ne marchera pas.

```
CREATE FUNCTION MesResultats( out Matiere id varchar(10), out Controle varchar,
 out Note decimal(4,2))
RETURNS SETOF RECORD
AS
$$
        SELECT m.Matiere id , c.Controle , n.Note
        FROM Etudiant e, Matiere m, Controle c, Notes n
                WHERE m.Matiere_id=c.Matiere_id
                 AND c.Contrôle_id =n.Contrôle_id
                 AND n.Etudiant_id =e.Etudiant_id
```

```
                AND e.Nom= session_user;
$$ language SQL
SECURITY DEFINER;
```

La procedure moderateur() permet seulement au modérateur (Claude) de la Base de Donnée de pouvoir faire des insertion ou suppression dans la table Notes

```
CREATE or replace FUNCTION moderateur()
        RETURNS TRIGGER AS
$$
        BEGIN
        IF current_user :: varchar =’Claude’ THEN
        IF TG_OP =’INSERT’ THEN
                Return NEW ;
        END IF ;
        IF TG_OP = ‘DELETE’ THEN
                return OLD ;
        END IF ;
END IF ;
RETURN NULL ;
END ;
$$ LANGUAGE plpgsql;

CREATE TRIGGER moderateur
        BEFORE
        INSERT or DELETE on Note
        for EACH ROW
        EXECUTE PROCEDURE moderateur();
```