SAE Exploitation d’une base de données (S204)

GROUPE INDRA
Professeur référent : Hocine Abir

Groupe : Iliasse Bellouch, Ahmed Elsaadawy, Niels Keethaponkalan, Azzedine Hatem

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

L'objectif de ce projet est de mettre en place un système de gestion des notes qui soit accessible, facilement consultable et pouvant être mis à jour.

Les objectifs spécifiques sont les suivants :

- Création d'une base de données adaptée à la configuration de l'IUT Villetaneuse Paris 13.
- Gestion de l'activité de la base de données avec possibilité de restreindre l'accès.
- Visualisation des informations pour analyse.
- Réalisation d'un travail technique complet incluant la conception, l'implémentation, l'administration et l'exploitation du système.

Le projet de base de données S204 est limité au contexte universitaire et bénéficiera uniquement aux étudiants, enseignants, responsables de matière et autres parties concernées. De plus, l'accès aux données sera restreint, avec seulement certaines personnes autorisées à les consulter.

Les logiciels et technologies utilisés pour ce projet seront SQL et PLpgSQL.

Pour mieux décrire les besoins et le projet, nous fournirons un modèle de données et un script de base de données afin de clarifier l'objectif final.

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.

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)
- Associé 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, Rattrapage, exam,Coefficient)
- A partir de l'étudiant, du module et du semestre sélectionnées on donne tous les informations correspondantes aux notes de cet élève dans le temps imparti du semestre.

Semestre (Num_Semestre, Semestre)
- Nous donne les semestres de l'année.

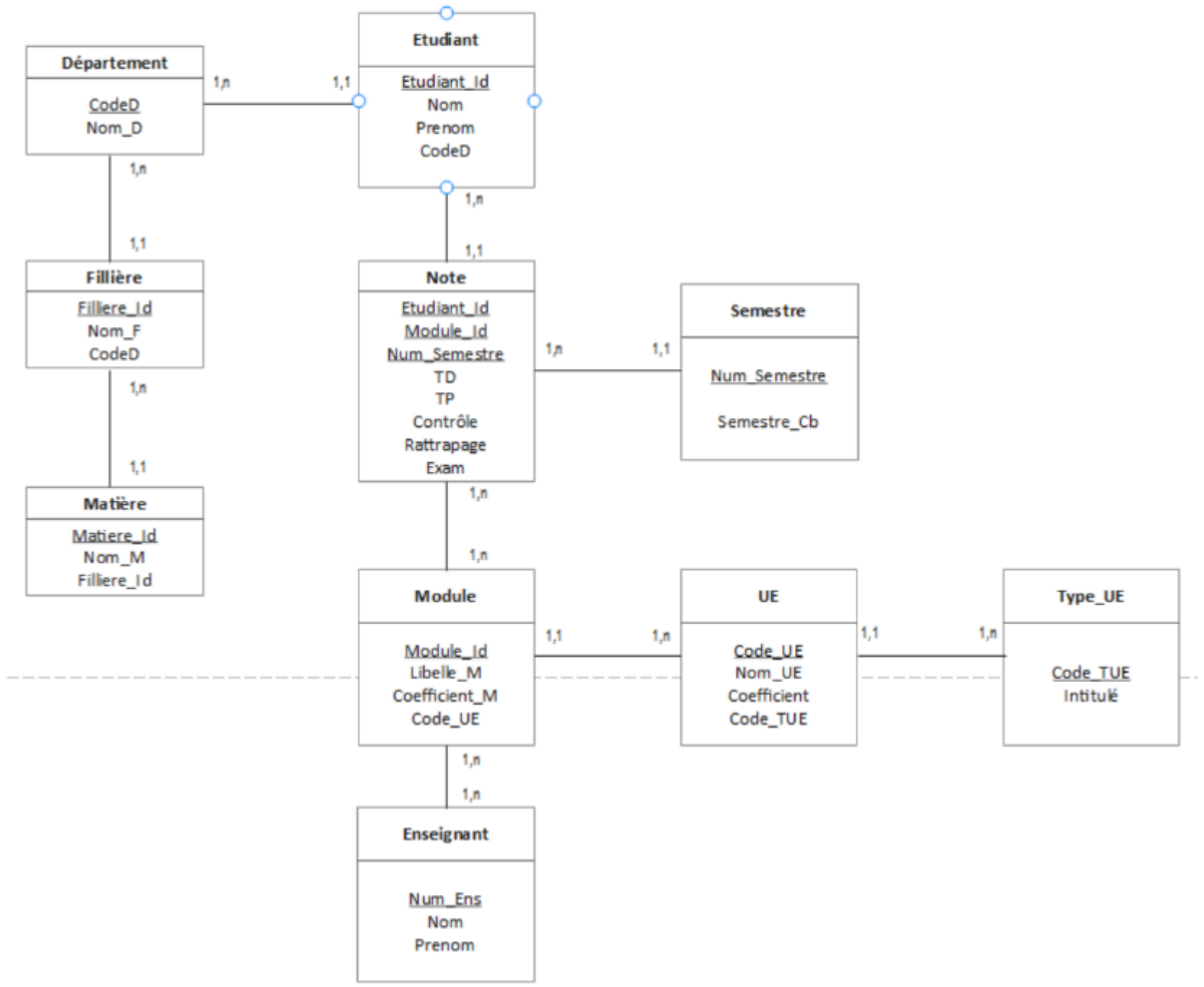
UE (Code_UE, Nom_UE, coefficient, Code_TUE)
- Retourne les informations sur un Type d'UE sélectionné.

Type_UE (Code_TUE, intitulé)
- Nous donne le type d'UE et son intitulé.

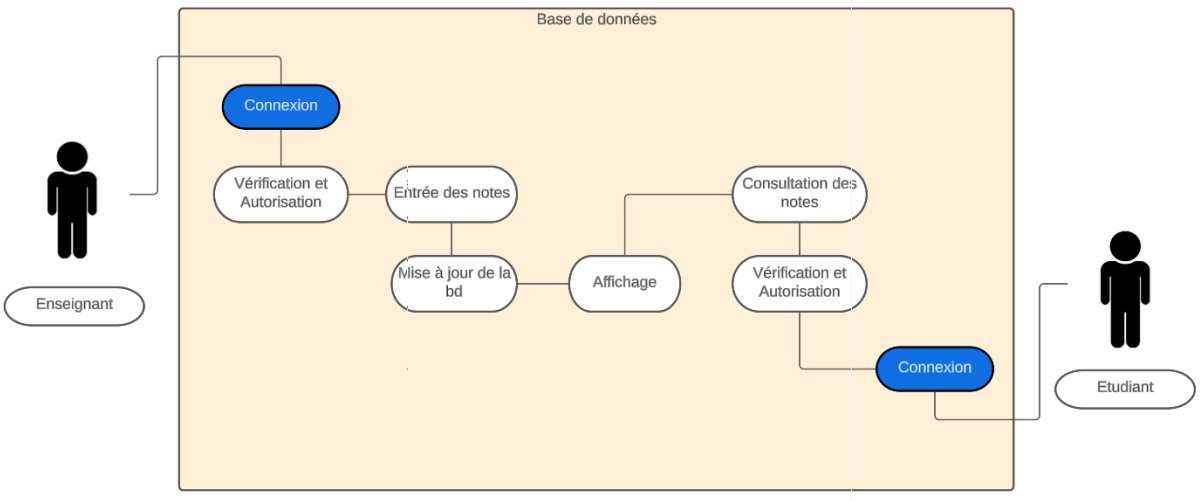
Il est évident que cette table contient de nombreuses informations nécessaires à la gestion de l'affichage des notes (Type UE, Semestre, Filières, Module). Ces données sont particulièrement intéressantes car elles facilitent la classification de la base de données et la rendent plus accessible.

Nous devons maintenant structurer ces données et les décrire en analysant le script, ce qui nous permettra de définir les restrictions de cette base de données.

Modèle de données

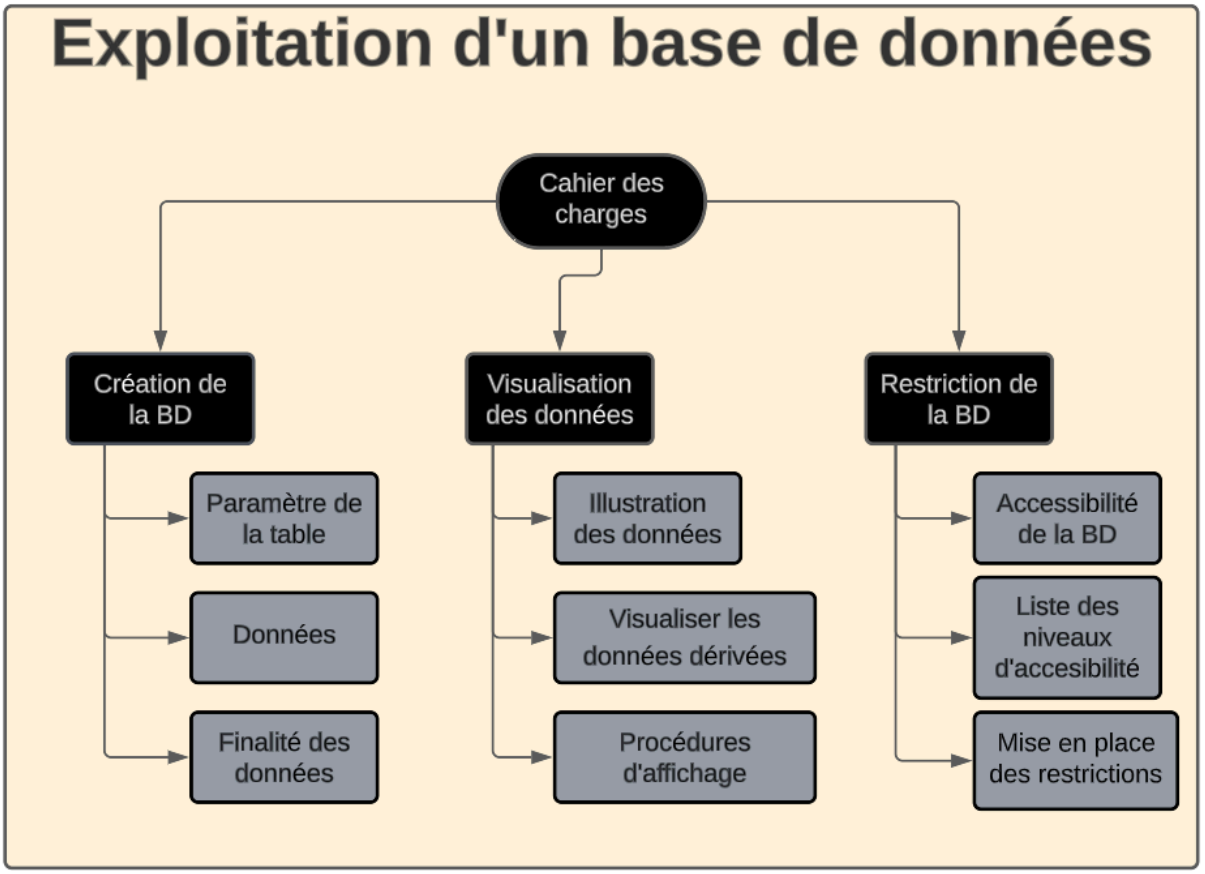


Lors de la production de notre base de données, nous devrons permettre à l'enseignant et à l'étudiant de répondre à ces besoins :



Dans ce schéma, l'enseignant est le seul à pouvoir entrer les notes qui seront par la suite affichées à l'élève.

Organigramme des tâches



Procédures stockées

Vous trouverez ci-dessous des exemples de procédures 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
$$
BEGIN
    IF EXISTS (SELECT 1 FROM Etudiant WHERE Etudiant_Id = numero) THEN
        RETURN QUERY SELECT Nom_E, TD, TP, Controle, Rattrapage, Exam FROM Note
WHERE Etudiant_Id = numero;
```

```
   ELSIF EXISTS (SELECT 1 FROM Enseignant WHERE Num_Ens = numero) THEN
      RETURN QUERY SELECT Nom_E, TD, TP, Controle, Rattrapage, Exam FROM Note;
   ELSE
      RAISE NOTICE 'Vous ne pouvez pas accéder aux notes';
   END IF;
END;

$$ LANGUAGE PL/pgSQL;


CREATE FUNCTION ajouter_note(N_E int, note int, variable varchar(20))
RETURNS void AS
$$
BEGIN
   IF EXISTS (SELECT 1 FROM Enseignant WHERE Num_Ens = N_E) AND note
BETWEEN 0 AND 20 THEN
      INSERT INTO Note (variable) VALUES (note);
   ELSE
      RAISE NOTICE 'Le numéro d''enseignant ou la note sont erronés';
   END IF;
END;

$$ LANGUAGE PL/pgSQL;



CREATE FUNCTION modifier_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 ajouter_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 ;
```

Avec cette procédure afficher_note, on utilise les IF-ELSIF-ELSE pour savoir si le numéro en paramètre correspond à un étudiant ou à un enseignant, et renvoie les notes si cela est vrai. Si le numéro ne correspond à aucun étudiant ni enseignant, un message d'information est envoyé.

Avec la procédure ajouter_note, on utilise des IF-ELSE pour vérifier si le numéro d'enseignant et la note sont valides avant d'insérer la note dans la table "Note". Si la condition n'est pas remplie, un message d'information est renvoyé.

Ensuite, la procédure modifier_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.

Enfin, la procédure ajout_semestre est semblable à ajout_note sauf qu'elle correspond aux semestres comme S1 et S2. Aussi, un semestre ne peut être inférieur ou égal à 0 et que les notes doivent être obligatoirement comprises entre 0 et 20.

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 en fonction de la 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
```

La vue notes_totales permet de visualiser les notes des étudiants par module.

```
CREATE VIEW notes_totales AS
SELECT Etudiant.Etudiant_Id, Etudiant.Nom_E, Etudiant.Prénom_E, Module.Libelle_M,
Note.TD, Note.TP, Note.Contrôle, Note.Rattrapage, Note.exam
FROM Note
INNER JOIN Etudiant ON Note.Etudiant_Id = Etudiant.Etudiant_Id
INNER JOIN Module ON Note.Module_Id = Module.Module_Id;
```

La procédure fil_etud affiche l'ensemble des noms
des étudiants qui font partie d'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;
```

Cette procédure etudiant_fil permet de récupérer la liste des étudiants d'une filière.

```
CREATE PROCEDURE etudiant_fil (IN filiereId INT)
BEGIN
   SELECT *
   FROM Etudiant
   WHERE CodeD = (
      SELECT CodeD
      FROM Filières
      WHERE Filière_Id = filiereId
   );
END
```

La procédure noteEtudP retourne 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 procédure donnée permet à chaque étudiant de visualiser
les données qui lui sont associées.

```
CREATE or REPLACE FUNCTION donnée( 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 autorise seulement les enseignants 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 procédure MesResultats permet à son utilisateur donc un 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 procédure moderateur() permet seulement au modérateur Michel de la Base de Donnée de pouvoir faire des insertions ou suppressions dans la table Notes.

```
CREATE or replace FUNCTION moderateur()
RETURNS TRIGGER AS
$$
```

```
BEGIN
IF current_user :: varchar =’Michel’ 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();
```