CAHIER DES CHARGES

Par le groupe numéro 4
-Kevin Postic (N° étudiant : 12104997)
-Pascal Zhan (N° étudiant : 12100404)
-Adriana-Mariana Fratila (N° étudiant : 12103419)

PRESENTATION DU BUT INFORMATIQUE :

La licence professionnelle de Bachelor Universitaire de Technologie informatique ou plus communément appelée le BUT informatique est une formation d'une durée de 3 ans qui permet d'obtenir le statut d'informaticien.
Cette formation est constituée de 2 semestres d'une durée totale de 500h chacun qui se divisent en 425h d'enseignement et 75h de travail encadré. Le semestre 1 est composé de 12 matières telles que les mathématiques discrètes, le développement d'interfaces web ou encore l'introduction à l'architecture des ordinateurs et le semestres 2 est composé de 14 matières telles que le développement orienté objets, l'introduction au services réseaux ou encore l'exploitations d'une base de données. Lors de chaque semestre les étudiants réaliseront 6 SAE ou "Situation d'Apprentissage et d'évaluation", ce sont des projets avec des consignes et exigences que nous pouvons avoir dans le monde professionnel, qui permettent d'appliquer les connaissances et compétences et ainsi professionnaliser. Afin de valider un semestre il faut avoir la moyenne dans 6 compétences.

PRÉSENTATION DU PROJET :

Ce projet à pour but de créer une base de données qui permet de gérer différents éléments de la formation du BUT informatique. Les étudiants doivent être en mesure de visualiser leurs notes,classement,nom de groupe, et leur emploi du temps (nom des matières et durée de chaque cours). Les enseignants pourront avoir accès à des informations et fonctionnalités supplémentaires telles que l'ajout et modification de notes.

LES ÉTAPES DU PROJET :

-Afin de créer la base de données nécessaires nous avons d'abord schématisé nos idées sous forme de diagrammes qui représentent nos différentes tables contenant les données en suivant un modèle de données qui permet de mettre en relation les tables.
-A partir de ce schéma on a réalisé le script SQL de cette base de données.
-Suite à cela nous avons rajouté des règles de gestion de données à notre script telles que des "procédures Triggers" qui permettent d'éviter l'ajout de

valeurs erronées sur la plateforme.

DESCRIPTION DES DONNÉES ET DES TABLES :

Pour créer les différentes tables on a réuni différentes informations :

-La table *“ Etudiant”* est constituée du “prénom” et “nom” des étudiants et de leur identifiants retranscrit sous le nom de “Etudiant_Id”
“Etudiant_Id” est un nombre (Int) et une clé primaire.
“prénom” et “nom” sont des caractères (Varchar)

-La table *"Groupe" est constituée* du nom du groupe et l’identifiant du groupe retranscrits respectivement sous les noms de “NomGrp” et “Grp_Id”.
“Grp_Id” est un nombre (Int) et une clé primaire
“NomGrp” correspond à des caractères (Varchar)

-La table *“Groupe_Etudiant”* fait référence aux tables *“Etudiants”* et *“Groupe”* et reprend les éléments “Groupe_Id” , “Etudiant_Id” qui sont des clés primaires faisant. Cette table indique le groupe X d’un étudiant Y.

-La table *“ Prof”* est constituée du “prénom” et “nom” des professeurs et leur identifiant retranscrit sous le nom de “Prof_Id”
“Prof_Id” est un nombre (Int) et une clé primaire.
“prénom” et “nom” sont des caractères (Varchar)

-La table *“Matiere”* est constituée du nom de la matière, le nombre d’heures de chaque matière et l’identifiant de ces matières retranscrits respectivement sous le nom de “Matiere”, “Volume_Matiere” et “Matiere_Id”
“Matiere” correspond à des caractères (Varchar)
“Volume_Matière” correspond à un nombre (Int)
“Matiere_Id” correspond à un nombre (Int) et est une clé primaire

-La table “*Competence” est* constitué du nom des compétences et leur identifiants retranscrits respectivement sous le nom de “Compétence” et “Compétence_Id”
“Compétence” correspond à des caractères (Varchar)
“Compétence_Id” correspond à un nombre (Int) et est une clé primaire

- La table *“Competence_Matiere”* fait référence aux tables *“Matiere”* , *“Compétence” , “Semestre”* et reprend les éléments “competence_Id” , “Matiere_Id” et “Semestre_id” qui sont des clés primaires, le coefficient des matières est retranscrit sous le de nom de “Coeff” et correspond a des caractères (Varchar). Cette table indique qu’une matière X fait partie d’une compétence Y.
-La table *“Controle”* est constituée du nom des contrôles et leur identifiant et du coefficient du contrôle retranscrits respectivement sous le nom de “controle”,

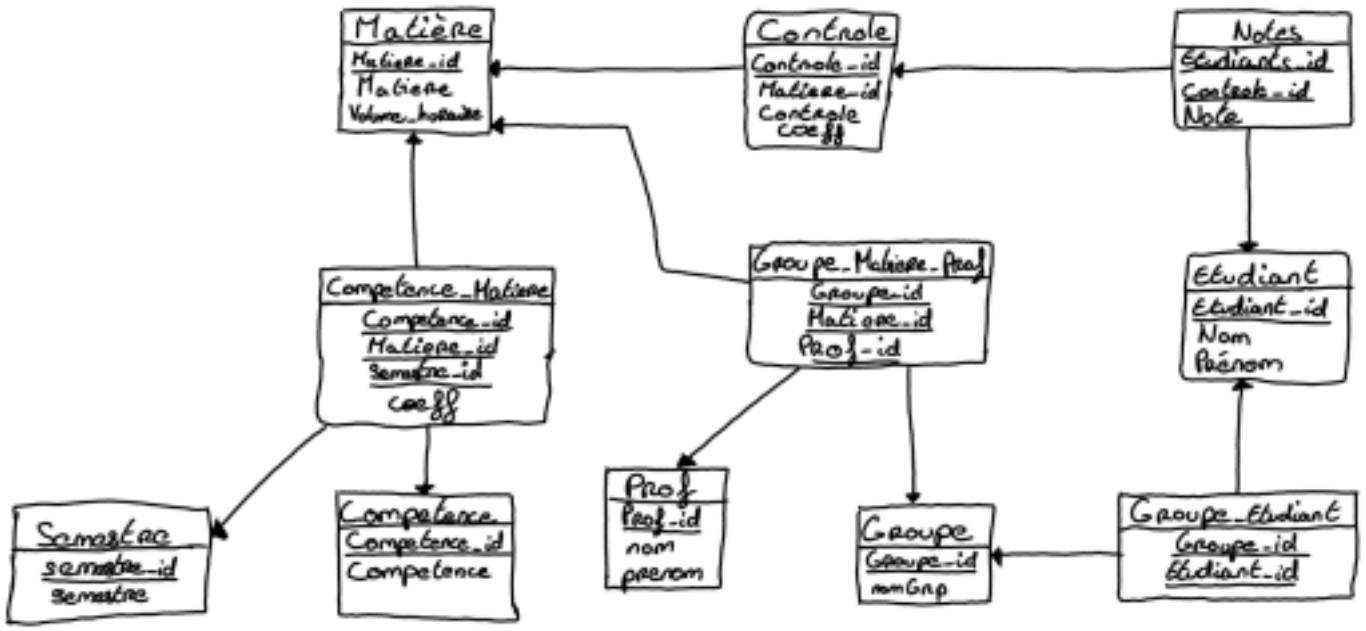
“Controle_Id” et “Coeff”, elle possède l’élément “Matiere_id“ qui fait référence à la table *“Matiere”*
*“*controle*”* correspond à des caractères (Varchar)
*“*Controle_id*”* correspond à un numéro (Int) et est une clé primaire
Coeff*”* correspond à un numéro (Int)

-La table *“Note”* est constituée de la note du contrôle retranscrit sous le nom de “Note” et possède les éléments “Etudiant_Id” qui fait référence àla table *“Etudiants”* et “Controle_id” qui fait référence à la table *“Controle”* , ces éléments sont des clés primaires.
Note correspond à un nombre decimal (4,2)
Cette table indique la note d’un étudiant X à un contrôle Y.

-la table *“Groupe_Matiere_Prof”* est constituée des éléments et clé primaires “Groupe_Id” faisant référence à la table *“Groupe”*, “Matiere_Id” faisant référence à la table *“Matiere”* et “Prof_id qui fait référence à la table *“Prof”*.
Cette table indique le groupe X d’un professeur Y enseignant la matière Z .

-la table *“Semestre”* est constituée du nom des semestres et de leur identifiant retranscrits respectivement sous le nom de “Semestre” et “Semestre_id” qui est une clé primaire.
Semestre correspond à des caractères (Varchar)
Semestre_id correspond à un numéro (int)

REPRÉSENTATION DES TABLES :



SCRIPT SQL :

```
CREATE TABLE IF NOT EXISTS ETUDIANT(
etudiant_id int primary key,
nom varchar(20),
prenom varchar(20)
);

CREATE TABLE IF NOT EXISTS MATIERE(
matiere_id int primary key,
matiere varchar(15),
volume_horaire int
);

CREATE TABLE if not exists COMPETENCE(
competence_id int primary key,
competence varchar(15)
);

create table if not exists SEMESTRE (
Semestre_id int primary key,
Semestre varchar(15)
);

CREATE TABLE if not exists PROF(
prof_id int primary key,
nom varchar(20),
prenom varchar(20)
);

CREATE TABLE if not exists CONTROLE( controle_id int primary
key, matiere_id int references MATIERE,
controle varchar(20),
coeff int
);

CREATE TABLE if not exists NOTES( etudiant_id int references
ETUDIANT, controle_id int references CONTROLE,
note decimal(4,2), primary key (etudiant_id,controle_id)
)
CREATE TABLE if not exists COMPETENCE_MATIERE(
competence_id int references COMPETENCE,
Semestre_id int references SEMESTRE ,
matiere_id int references MATIERE,
coeff int,
primary key (competence_id,Semestre_id,matiere_id)
);
```

```
CREATE TABLE if not exists GROUPE(
groupe_id int primary key,
nomgrp varchar(15)
);

CREATE TABLE if not exists GROUPE_ETUDIANT(
groupe_id int references GROUPE,
etudiant_id int references ETUDIANT,primary
key(groupe_id,etudiant_id) );

CREATE TABLE if not exists
GROUPE_MATIERE_PROF( groupe_id int references
GROUPE,
matiere_id int references MATIERE,
prof_id int references PROF,primary
key(groupe_id,matiere_id,prof_id) );


CREATE or replace FUNCTION check_note()--verifier la note
  returns TRIGGER AS
$$
  Begin
        IF NEW.note < 0 or NEW.note > 20
              then
                    return null;
        end IF;
        RETURN NEW;
        end;

$$language plpgsql;
DROP TRIGGER IF EXISTS tmaj_note on notes;
CREATE TRIGGER tmaj_note --mettre à jour
        BEFORE
        UPDATE ON notes
              FOR EACH ROW
              EXECUTE PROCEDURE check_note();

DROP TRIGGER IF EXISTS tadd_note on notes;
CREATE TRIGGER tadd_note --ajouter une note
        BEFORE
        INSERT ON notes
              FOR EACH ROW
              EXECUTE PROCEDURE check_note();

CREATE or replace FUNCTION check_volH()--verifier le volume horaire
  returns TRIGGER AS
```

```
$$
  Begin
        IF NEW.volume_horaire < 0
              then
                    return null;
        end IF;
        RETURN NEW;
        end;

$$language plpgsql;

DROP TRIGGER IF EXISTS tmaj_volH on
MATIERE; CREATE TRIGGER tmaj_volH --mettre à
jour
        BEFORE
        UPDATE ON MATIERE
              FOR EACH ROW
              EXECUTE PROCEDURE check_volH();

DROP TRIGGER IF EXISTS tadd_volH on
MATIERE; CREATE TRIGGER tadd_volH --ajouter
        BEFORE
        INSERT ON MATIERE
              FOR EACH ROW
              EXECUTE PROCEDURE check_volH();

CREATE or replace FUNCTION check_coeff()--verifier le
  coeff returns TRIGGER AS
$$
  Begin
        IF NEW.coeff < 0
              then
                    return null;
        end IF;
        RETURN NEW;
        end;

$$language plpgsql;

DROP TRIGGER IF EXISTS tmaj_coeff on
COMPETENCE_MATIERE; CREATE TRIGGER tmaj_coeff --mettre à
jour
        BEFORE
        UPDATE ON COMPETENCE_MATIERE
              FOR EACH ROW
              EXECUTE PROCEDURE check_coeff();
```

```
DROP TRIGGER IF EXISTS tadd_coeff on
COMPETENCE_MATIERE; CREATE TRIGGER tadd_coeff--ajouter
        BEFORE
        INSERT ON COMPETENCE_MATIERE
                FOR EACH ROW
                EXECUTE PROCEDURE check_coeff();

DROP TRIGGER IF EXISTS tmaj_coeff on
CONTROLE; CREATE TRIGGER tmaj_coeff --mettre à
jour BEFORE
        UPDATE ON CONTROLE
                FOR EACH ROW
                EXECUTE PROCEDURE check_coeff();

DROP TRIGGER IF EXISTS tadd_coeff on
CONTROLE; CREATE TRIGGER tadd_coeff--ajouter
        BEFORE
        INSERT ON CONTROLE
                FOR EACH ROW
                EXECUTE PROCEDURE check_coeff();
```

Partie FIN

Pour la base de données, nous avons décidé de créer les vues et règles d'accès suivant:

1. Relevé de notes des étudiants
2. Relevé de notes d'un groupe
3. Classement par UE
4. Classement général
5. Moyenne max et min de chaque contrôle
6. Un étudiant ne peut consulter que ses propres notes
7. Un enseignant doit pouvoir saisir les notes de ses contrôles
8. Un étudiant ne peut modifier ses notes
9. Un enseignant ne peut modifier que les notes des étudiants de son groupe et de sa matière
10. Un enseignant peut consulter le classement des étudiants dans la promotion
11. Un étudiant ne peut voir que son rang et non le classement général

Voici le code :

```
CREATE OR REPLACE VIEW Moyennes_matiere
AS
        SELECT e.Etudiant_id, Nom, Prenom, 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;
```

```
CREATE OR REPLACE VIEW Notes_etudiant
AS
	SELECT e.Etudiant_id, Nom, Prenom, 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;

CREATE OR REPLACE VIEW Notes_Groupe
AS
	SELECT g.groupe_id, g.nomgrp, m.Matiere_id, avg(Note) as moyenne
	FROM GROUPE g, GROUPE_ETUDIANT ge, Etudiant e, Matiere m, Controle c, Notes n
			WHERE g.groupe_id = ge.groupe_id
			AND ge.etudiant_id = e.etudiant_id
			AND m.Matiere_id=c.Matiere_id
			AND c.Controle_id =n.Controle_id
			AND n.Etudiant_id =e.Etudiant_id
			GROUP BY g.groupe_id, g.nomgrp, m.Matiere_id;

CREATE OR REPLACE VIEW Note_Max_Min
AS
	SELECT Controle.controle,MIN(Note),MAX(Note)
	FROM Controle NATURAL JOIN Notes
		GROUP BY Controle;

CREATE OR REPLACE VIEW classement_UE
AS
	SELECT c.competence, e.etudiant_id, e.Nom, e.Prenom, avg(Note) as Moyenne,
COUNT(Moyenne) as Rang
	FROM Etudiant e, Notes n, Controle ct, Matiere m, Competence_Matiere cm
		, COMPETENCE c
		WHERE n.Etudiant_id =e.Etudiant_id
		AND ct.Controle_id =n.Controle_id
		AND m.Matiere_id=ct.Matiere_id
		AND m.Matiere_id=cm.matiere_id
		AND cm.competence_id = c.competence_id
		GROUP BY c.competence,e.etudiant_id, e.Nom, e.Prenom
		ORDER BY Moyenne;

CREATE OR REPLACE VIEW classement
AS
	SELECT e.Etudiant_id, Nom, Prenom, m.Matiere_id, avg(Note) as moyenne, COUNT(Moyenne)
as Rang
		FROM Etudiant e, Notes n, Controle c, Matiere m
```

```
                        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
                        ORDER BY Moyenne;


CREATE OR REPLACE FUNCTION check_prof()
 returns TRIGGER AS
$$
DECLARE
        Pnom Varchar(50);
BEGIN
        SELECT nom INTO Pnom
                FROM PROF;
        IF Pnom != session_user THEN
                RETURN null;
        END IF;
        return new;
end;
$$ language plpgsql;

DROP TRIGGER IF EXISTS tmaj_note on notes;
CREATE TRIGGER tmaj_note --inserer une valeur
        BEFORE
        INSERT ON notes
        FOR EACH ROW
                EXECUTE PROCEDURE check_prof();


CREATE OR REPLACE FUNCTION check_profDeEtudiant()
 returns trigger as
$$
DECLARE
        P_id int;
        Pnom varchar(50);
BEGIN
        SELECT p.prof_id into P_id
                FROM PROF p, GROUPE_MATIERE_PROF gmp, GROUPE g, GROUPE_ETUDIANT
ge, Etudiant e, Notes n
                        WHERE p.prof_id = gmp.prof_id
                        AND gmp.groupe_id = g.groupe_id
                        AND g.groupe_id = ge.groupe_id
                        AND ge.etudiant_id = e.etudiant_id
                        AND e.etudiant_id = n.etudiant_id
                        AND n.etudiant_id = OLD.etudiant_id;
```

```
        SELECT nom into Pnom
                FROM PROF WHERE P_id = prof_id;

        IF Pnom != session_user THEN
                return null;
        end if;

        if  TG_OP='UPDATE' THEN
                return NEW;
        end if;
        if  TG_OP='DELETE' THEN
                return OLD;
        END IF;
END;
$$ language plpgsql;

DROP TRIGGER IF EXISTS note_prof on notes;
CREATE TRIGGER note_prof
        BEFORE
        DELETE or UPDATE on notes
        for EACH ROW
                EXECUTE PROCEDURE check_profDeEtudiant();

CREATE or replace FUNCTION MesResultats( out Matiere_id int, out Controle varchar(20),
out Note float)
returns setof record
as
$$
        SELECT m.Matiere_id , c.Controle , n.Note
                FROM Etudiant e, Matiere m, Controle c, Notes n
                        WHERE m.Matiere_id=c.Matiere_id
                        AND c.Controle_id =n.Controle_id
                        AND n.Etudiant_id =e.Etudiant_id
                        AND e.Nom= session_user;
$$ language SQL
        SECURITY DEFINER;


CREATE or replace FUNCTION MonRang(out rang int)
returns int
as
$$
DECLARE
        n varchar(50);
BEGIN
```

```
	SELECT nom into n
		FROM Etudiant;
	IF session_user != n THEN
		return;
	END IF;
	SELECT classement.rang into rang
		FROM classement
			Where nom = session_user;
END;
$$ language plpgsql
	SECURITY DEFINER;

CREATE or replace FUNCTION Rang(out id int, out Nom varchar(20), out Prenom varchar(20), out
moyenne float, out rang int)
returns setof RECORD
as
$$
DECLARE
	n varchar(50);
BEGIN
	SELECT nom into n
		FROM PROF;
	IF session_user != n THEN
		return;
	END IF;
	return QUERY SELECT * FROM classement;
END;
$$ language plpgsql
	SECURITY DEFINER;
```