NAIM Charlotte et KOFFI Grace BUT Informatique, groupe ZEUS

Base de donnée et langage SQL

Sommaire

I. Modélisation de Données

1. Cahier des charges

Le but du projet est de créer une base de données de gestion des notes et des étudiants en BUT avec la configuration de l'IUT de Villetaneuse.
Cette base de données est accessible aux étudiants, aux enseignants ainsi qu'aux responsables de matière avec des restrictions différentes pour chacun.

Pour rappel, il y a 2 semestres dans une année. Il y a différents modules avec dans chaque module plusieurs ressources (matières). Chaque ressource peut recevoir plusieurs notes de contrôle.
De plus, un enseignant peut être responsable d'une ou plusieurs matières ou être responsable d'un groupe.

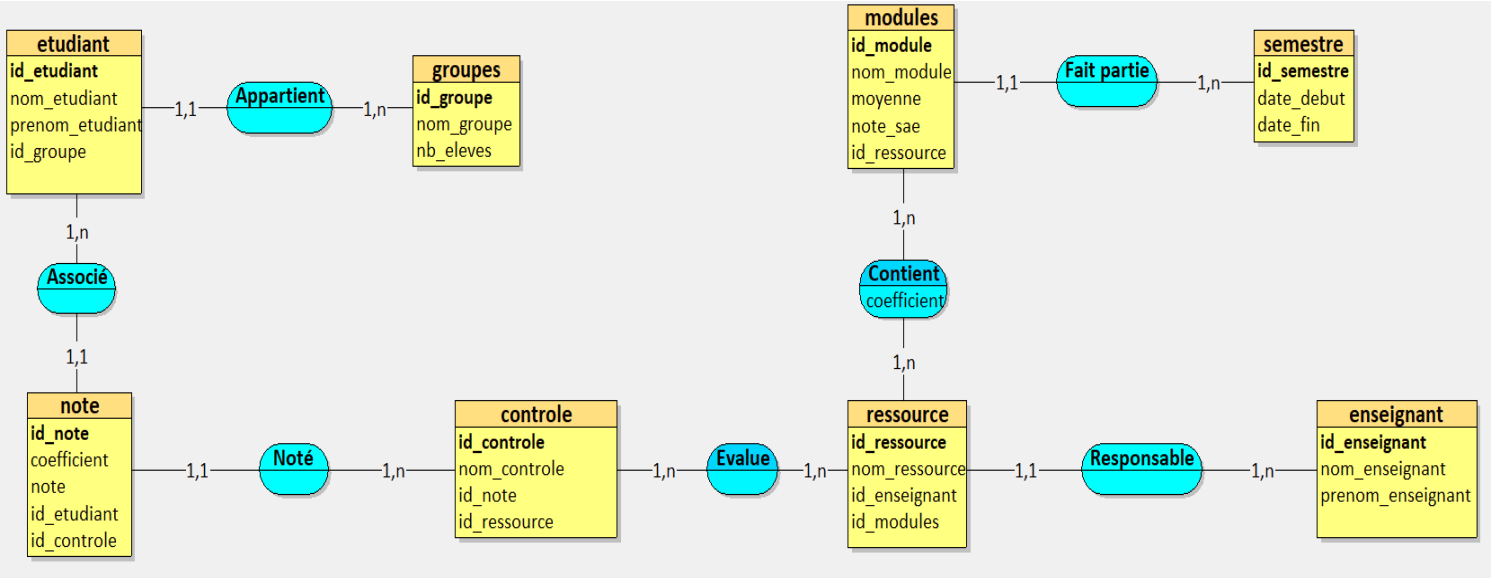
Pour ce projet, les logiciels utilisés sont PostgreSQL et Looping.

Cette SAÉ permet une première approche complète des aspects de conception, implémentation, administration et exploitation d'une base de données.

Différents objectifs:
- mise en place d'une base de données
- gérer la base de donnée et les restrictions
- visualisation des informations pour faire l'analyse

8 tables:
Groupes
Etudiant
Semestre
Ressource
Contrôle
Enseignant
Note
SAE
Modules

2. Modèles de données



3. Règle de gestion et mise en oeuvre par des procédure stockées

Ajouter un étudiant

```
CREATE FUNCTION ajouter_etudiant(
    nom_etudiant VARCHAR(255),
    prenom_etudiant VARCHAR(255),
    id_groupe INT,
    id_etudiant INT
)
RETURNS VOID AS $$
BEGIN
    INSERT INTO etudiant (id_etudiant, nom_etudiant, prenom_etudiant, id_groupe)
    VALUES (id_etudiant, nom_etudiant, prenom_etudiant, id_groupe);
END;
$$ LANGUAGE plpgsql;
```

Supprimer un enseignant

```
CREATE FUNCTION supprimer_enseignant(id_enseignant INT)
RETURNS VOID AS $$
BEGIN
    DELETE FROM enseignant WHERE id_enseignant = id_enseignant;
END;
$$ LANGUAGE plpgsql;
```

Modifier un module

```
CREATE FUNCTION modifier_module(
    id_module INT,
    nom_module VARCHAR(255),
    moyenne FLOAT,
    note_sae VARCHAR(255),
    id_ressource INT
)
RETURNS VOID AS $$
BEGIN
    UPDATE modules
    SET nom_module = nom_module, moyenne = moyenne, note_sae = note_sae,
id_ressource = id_ressource
    WHERE id_module = id_module;
END;
$$ LANGUAGE plpgsql;
```

4. Script de création de la base de données

```
CREATE TABLE groupes (
    id_groupe INT PRIMARY KEY,
    nom_groupe VARCHAR(255),
    nb_eleves INT
);

CREATE TABLE etudiant (
    id_etudiant INT PRIMARY KEY,
    nom_etudiant VARCHAR(255) NOT NULL,
    prenom_etudiant VARCHAR(255) NOT NULL,
    id_groupe INT NOT NULL,
    FOREIGN KEY (id_groupe) REFERENCES groupes(id_groupe)
);

CREATE TABLE note (
    id_note INT PRIMARY KEY,
    coefficient INT,
    note FLOAT,
    id_etudiant INT REFERENCES etudiant(id_etudiant),
    id_controle INT REFERENCES controle(id_controle),
);
```

```
CREATE TABLE controle (
   id_controle INT PRIMARY KEY,
   nom_controle VARCHAR(255),
   id_ressource VARCHAR(5),
   FOREIGN KEY (id_ressource) REFERENCES ressource(id_ressource)
);

CREATE TABLE ressource (
   id_ressource VARCHAR(5) PRIMARY KEY,
   nom_ressource VARCHAR(255),
   id_enseignant INT,
   FOREIGN KEY (id_enseignant) REFERENCES enseignant(id_enseignant)
   id_module VARCHAR(5),
   FOREIGN KEY (id_module) REFERENCES modules(id_module)
);

CREATE TABLE enseignant (
   id_enseignant INT PRIMARY KEY,
   nom_enseignant VARCHAR(255),
   prenom_enseignant VARCHAR(255)
);

CREATE TABLE modules (
   id_module VARCHAR(4) PRIMARY KEY,
   nom_module VARCHAR(255),
   moyenne INT,
   note_sae FLOAT,
   id_semestre INT,
   FOREIGN KEY (id_semestre) REFERENCES semestre(id_semestre),
);

CREATE TABLE semestre (
   id_semestre INT PRIMARY KEY,
   date_debut DATE,
   date_fin DATE
);
```

II. Visualisation de Données

1. Définir un ensemble de données dérivées à visualiser

- Récupérer la liste des modules des ressources
- Récupérer la liste des étudiants ayant eu une certaines note dans les ressources dont l'enseignant est responsable
- Récupérer la liste des contrôles des ressources dont un enseignant est responsable
- Retourne la liste des coefficients des différents contrôles de chaque ressources d'un semestre

2. Décrire des procédures, vues ou vues matérialisées pour accéder à ces données

Récupérer la liste des modules des ressources

Vues:

```
CREATE VIEW vue_liste_modules_ressource AS
SELECT nom_module
FROM liste_modules_ressource(<id_ressource>);
```

Procédure:

```
CREATE FUNCTION liste_modules_ressource(id_ressource INT)
RETURNS TABLE (
   nom_module VARCHAR(255)
)
AS $$
BEGIN
   RETURN QUERY
   SELECT m.nom_module
   FROM modules m
   INNER JOIN ressource r ON m.id_module = r.id_module
   WHERE r.id_ressource = id_ressource;
END;
$$ LANGUAGE plpgsql;
```

Appel de la fonction:

*SELECT * FROM* liste_modules_ressource*(id_ressource);*

Récupérer la liste des étudiants ayant eu une certaines note dans les ressources dont l'enseignant est responsable

Vues:

```
CREATE VIEW vue_liste_etudiants_note
AS SELECT nom_etudiant, prenom_etudiant, note
FROM liste_etudiants_note(id_enseignant INT, Note FLOAT);
```

Procédure stockées:

```
CREATE FUNCTION liste_etudiants_note(id_enseignant INT, Note FLOAT)
RETURNS TABLE (
   nom_etudiant VARCHAR(255),
   prenom_etudiant VARCHAR(255),
   note FLOAT
)
AS $$
BEGIN
   RETURN QUERY
   SELECT e.nom_etudiant, e.prenom_etudiant, n.note
   FROM etudiant e
   INNER JOIN note n ON e.id_etudiant = n.id_etudiant
   INNER JOIN ressource r ON n.id_ressource = r.id_ressource
   INNER JOIN enseignant ens ON r.id_enseignant = ens.id_enseignant
   WHERE n.note = Note AND ens.id_enseignant = id_enseignant;
END;
$$ LANGUAGE plpgsql;
```

Appel de la fonction:

```
SELECT * FROM liste_etudiants_note;
```

Récupérer la liste des contrôles des ressources dont un enseignant est responsable

Vue:
*CREATE VIEW vue*_controles_enseignant *AS*
SELECT nom_controle, ressource_concernee
FROM controles_enseignant*(id_enseignant);*

Procédure stockées:

```
CREATE FUNCTION controles_enseignant(id_enseignant INT)
RETURNS TABLE (
   nom_controle VARCHAR(255),
   ressource_concernee VARCHAR(255)
)
AS $$
BEGIN
   RETURN QUERY
   SELECT c.nom_controle, c.ressource_concernee
   FROM controle c
   INNER JOIN ressource r ON c.id_controle = r.id_controle
   WHERE r.id_enseignant = id_enseignant;
END;
$$ LANGUAGE plpgsql;
```

Appel de la fonction:

```
SELECT * FROM controles_enseignant(id_enseignant);
```

Retourne la liste des coefficients des différents contrôles de chaque ressources d'un semestre

Vues:

```
CREATE VIEW vue_coefficients_controles_semestre AS
SELECT nom_ressource, nom_controle, coefficient
FROM coefficients_controles_semestre(id_semestre);
```

Procédure stockées:

```
CREATE FUNCTION coefficients_controles_semestre(id_semestre INT)
RETURNS TABLE (
    nom_ressource VARCHAR(255),
    nom_controle VARCHAR(255),
    coefficient INT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT r.nom_ressource, c.nom_controle, c.coefficient
    FROM ressource r
    INNER JOIN controle c ON r.id_ressource = c.id_ressource
    WHERE r.id_semestre = id_semestre;
END;
$$ LANGUAGE plpgsql;
```

Appel de la fonction:

```
SELECT * FROM coefficients_controles_semestre;
```

III. Restrictions d'accès aux Données

1. Définir des règles d’accès aux données

- Un enseignant ne peut pas noter une matière dont il n’est pas responsable
- Un étudiant ne peut consulter que ses propres notes
- Une nouvelle note ne peut pas être inférieur à une ancienne note

2. Décrire des procédures ou vues pour mettre en oeuvre ces règles

Un enseignant ne peut pas noter une matière dont il n’est pas responsable

```
CREATE ROLE enseignant;
GRANT INSERT, UPDATE ON notes TO enseignant;
ALTER USER <nom_utilisateur> IN ROLE enseignant;
CREATE POLICY restriction_notes ON notes
  FOR INSERT, UPDATE
  USING (matiere_id IN (SELECT matiere_id FROM enseignant_matiere WHERE
enseignant_id = current_user));
```

Un étudiant ne peut consulter que ses propres notes

```
CREATE ROLE etudiant;
GRANT CREATE MATERIALIZED VIEW ON notes TO etudiant;
ALTER USER <nom_utilisateur> IN ROLE etudiant;
CREATE POLICY restriction_notes_students ON notes
  FOR SELECT
  USING (etudiant_id = current_user);
```

Une nouvelle note ne peut pas être inférieur à une ancienne note

```
CREATE FUNCTION new_notes() returns TRIGGER
AS $$
BEGIN
        IF NEW.note < OLD.note
        THEN
                RETURN NULL;
        END IF;
        RETURN NEW;
END;
$$ language plpgsql;
```