Cahier des charges
de la Base de données SAE 204

Contexte et définition du projet

Le projet de base de données de la SAE 204 a pour but de mettre en place une base de données dans le contexte d'une gestion des notes des étudiants en BUT. La base de données devra comporter les informations relatives aux étudiants,enseignants, matières et notes .Elle devra permettre la mise en œuvre de données dérivées comme un relevé de note ou un bilan. Elle devra disposer de différentes permissions accordées ou non en fonction du rôle des personnes soit: étudiantes , enseignantes ou responsable de matière .

Objectif du projet

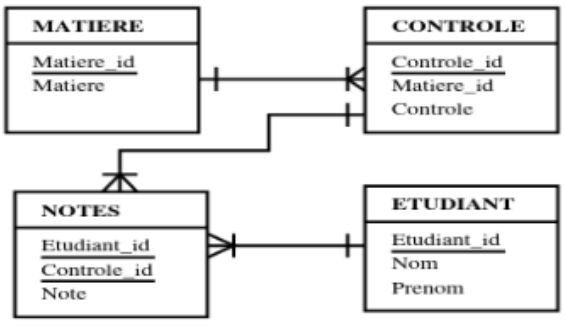
L'objectif de ce projet est de faciliter la gestion des notes des étudiants en BUT c'est-à- dire le stockage et l'organisation des notes grâce à une base de données efficace et sécurisée pour toutes les personnes , étudiantes ou enseignantes ,qui pourront accéder à cette base de données.

Ressources initiales

Nous possédons comme ressources initiales :

1. un exemple de modèle de données

Exemple d'illustration :



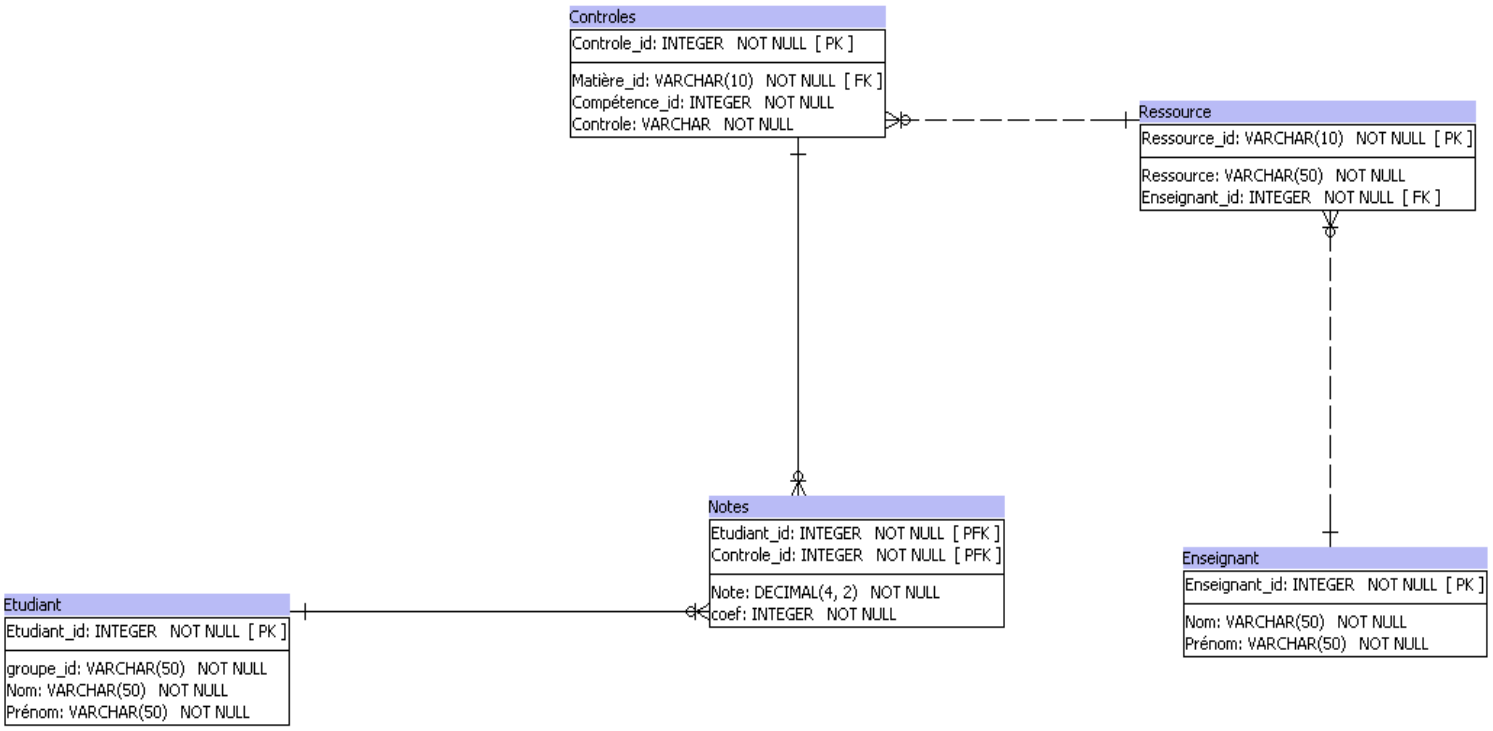
Schema Canonique

2. Un exemple de script de création de base de données

```
CREATE TABLE Etudiant
  (
        Etudiant_id    serial
                       primary key,

        Nom varchar(50),
        Prenom varchar(50)
  );

CREATE TABLE Matiere
  (
        Matiere_id  varchar(10)
                    primary key,
        Matiere     varchar
  );

CREATE TABLE Controle
  (
        Controle_id     serial
                        primary key,
        Matiere_id  varchar(10)
                        references  Matiere( Matiere_id ),
        Controle        varchar
  );

CREATE TABLE Notes
  (
        Etudiant_id     int
                        references  Etudiant( Etudiant_id ),
        Controle_id     int
                        references  Controle( Controle_id ),
        note            decimal(4,2),
           primary key(Etudiant_id, Controle_id )
  );
```

Exigences fonctionnelles des besoins

Le modèle de données :s



Gestion des étudiants :

- Enregistrer les informations des étudiants, y compris leur nom, prénom, numéro d'étudiant, adresse, etc.
- Permettre la création, la mise à jour et la suppression des profils étudiants.
- Associer chaque étudiant à une classe spécifique (INDRA, TLALOC, ZEUS, etc.).
- Permettre aux étudiants de consulter leurs propres informations personnelles.

Gestion des enseignants :

- Enregistrer les informations des enseignants, y compris leur nom, prénom et id.
- Associer chaque enseignant à une ressource.
- Permettre aux enseignants de consulter les informations relatives à leur matière spécifique.

Gestion des matières :

- Enregistrer les informations des matières, y compris leur nom, code.
- Permettre la création, la mise à jour et la suppression des matières.

Gestion des notes :

- Enregistrer les notes des étudiants pour chaque contrôle.
- Permettre aux enseignants de saisir les notes des étudiants.
- Permettre aux étudiants de consulter leurs propres notes.
- Calculer automatiquement la moyenne de chaque étudiant pour chaque matière.

- Calculer automatiquement la moyenne de chaque classe pour chaque matière.

Exigences non fonctionnelles des besoins

Techniques :

Les scripts et la base de données seront fait et codées en langage mySQL et/ou pl/pgsql,

Sécurité :

Les personnes ayant accès à la base de données devront avoir différentes permissions d'accès ou non aux procédures et vues qui représentent des rôles/groupes.

Les règles d'accès au données devront donc respecter une division en 4 groupes , un groupe "étudiant" qui représentera ce que l'étudiant seul pourra voir , un autre groupe "professeurs" qui représente ce que les professeurs pourront voirs, le 3ème groupe est un groupe qui englobe tout le monde puis un derniers groupe qui concerne les différentes "classes" c'est à dire INDRA , TLALOC, ZEUS etc.

Le groupe "étudiant" devra avoir la capacité de voir sa propre moyenne et ses propres relevés de notes mais pas celles des autres étudiants.
Le groupe des "professeurs" pourra voir les moyennes de tous les élèves , le relevé de note des groupes et le relevé de note de chaque étudiant.
Le groupe "classe" pourra voir le relevé de note de son propre groupe.
Le groupe "tout le monde" aura accès à la note maximum de chaque matière parmi tous les élèves ainsi que la minimum mais aussi la moyenne de chaque classe.

Chacun des groupes est attribué selon si on est un professeur ou étudiant et à quelle classe appartient t-on .
Il faut savoir que le groupe “tout le monde” est attribué logiquement à tout le monde c’est à dire que c’est le rôle par défaut et que chaque personne sera dans ce groupe.
Et une personne peut avoir plusieurs groupes comme l’étudiant qui est à la fois étudiant , classe et tout le monde.

Performance :
La base de données devra être accessible par toutes les personnes travaillant ou étudiant à l'IUT (sauf technicien de surfaces et approvisionneur de distributeurs automatique) et elle devra fonctionner 24h sur 24 et tous les jours de la semaines , elle doit donc être un minimum optimisé pour maintenir un service fluide et fiable

Délais du projet

Le projet est à rendre impérativement pour le mardi 23 mai 2023.

Perspective d’évolution du projet

Tout d’abord on pourrait faire Intégration des fonctionnalités de tracking et de notification : Pour améliorer la communication et le suivi, il serait intéressant d'ajouter des fonctionnalités de tracking et de notification. Par exemple, les étudiants peuvent recevoir des notifications automatiques leur rappelant les dates d'échéance des évaluations, les enseignants peuvent être informés des changements dans les notes des étudiants et les responsables de matière peuvent être informés des statistiques de performance de leur matière.

On pourrait aussi rajouter À mesure que votre base de données se développe et que de plus en plus de personnes y accèdent, il est important de surveiller et d'optimiser les performances. Cela peut impliquer l'optimisation des requêtes, l'indexation appropriée des tables, la mise en cache des résultats fréquemment consultés et la mise à l'échelle du système pour gérer une charge croissante.

On pourrait aussi étendre la base de données aux autres institutions au sein de l’université en les liant aux autres IUT de paris pour permettre d’obtenir des statistiques plus représentative du niveau des élèves et de la productivité de ceci ainsi que des professeurs .

SCRIPT Finale du projet

```
-----------------------------------------------------------------
----------1-----------------------------------------------------
--------
CREATE TABLE Etudiant (
    Etudiant_id serial primary key,
    groupe_id VARCHAR(10),
    Nom varchar(50),
    Prenom varchar(50)
);

CREATE TABLE Enseignant (
    Enseignant_id serial PRIMARY KEY,
    Nom varchar(50),
    Prenom varchar(50)
);

CREATE TABLE Ressource (
    Ressource_id varchar(10) primary key,
    Ressource varchar(50),
    enseignant_id int REFERENCES Enseignant(Enseignant_id)
);

CREATE TABLE Controle (
    Controle_id serial primary key,
    Matiere_id varchar(10) REFERENCES Ressource(Ressource_id),
```

```
    Controle varchar(50),
    competence_id int
);

CREATE TABLE Notes (
    Etudiant_id int REFERENCES Etudiant(Etudiant_id),
    Controle_id int REFERENCES Controle(Controle_id),
    note decimal(4, 2),
    coef int,
    primary key (Etudiant_id, Controle_id)
);
-----------------------------------------------------------------
----------------------3-----------------------------------------
-----------------------------------
CREATE VIEW VueMoyennesEtudiants AS
SELECT E.Etudiant_id, E.Nom, E.Prenom, AVG(N.note * N.coef) AS
Moyenne
FROM Etudiant E
JOIN Notes N ON E.Etudiant_id = N.Etudiant_id
GROUP BY E.Etudiant_id, E.Nom, E.Prenom;

CREATE VIEW Classement_etudiants AS
SELECT Etudiant.Etudiant_id, Etudiant.Nom, Etudiant.Prenom,
Moyenne_notes.Moyenne,
       RANK() OVER (ORDER BY Moyenne_notes.Moyenne DESC) AS Rang
FROM VueMoyennesEtudiants Moyenne_notes
JOIN Etudiant ON Etudiant.Etudiant_id = Moyenne_notes.Etudiant_id;

CREATE VIEW VueMoyennesGroupes AS
SELECT E.groupe_id AS NomGroupe, AVG(N.note * N.coef) AS Moyenne
FROM Etudiant E
JOIN Notes N ON E.Etudiant_id = N.Etudiant_id
GROUP BY E.groupe_id;

CREATE VIEW VueMoyennesCompetences AS
SELECT C.competence_id, R.Ressource_id, R.Ressource, AVG(N.note *
N.coef) AS Moyenne
FROM Controle C
JOIN Ressource R ON C.Matiere_id = R.Ressource_id
JOIN Notes N ON C.Controle_id = N.Controle_id
GROUP BY C.competence_id, R.Ressource_id, R.Ressource;

CREATE VIEW VueMoyennesMatieres AS
SELECT R.Ressource_id, R.Ressource, AVG(N.note * N.coef) AS
Moyenne
FROM Ressource R
JOIN Controle C ON R.Ressource_id = C.Matiere_id
JOIN Notes N ON C.Controle_id = N.Controle_id
```

```
GROUP BY R.Ressource_id, R.Ressource;

CREATE VIEW VueNotesMatiereControle AS
SELECT R.Ressource_id, R.Ressource, C.Controle_id, C.Controle,
N.note
FROM Ressource R
JOIN Controle C ON R.Ressource_id = C.Matiere_id
JOIN Notes N ON C.Controle_id = N.Controle_id;

CREATE VIEW VueResultatsEtudiants AS
SELECT E.Etudiant_id, E.Nom, E.Prenom, R.Ressource, C.Controle,
N.note
FROM Etudiant E
JOIN Notes N ON E.Etudiant_id = N.Etudiant_id
JOIN Controle C ON N.Controle_id = C.Controle_id
JOIN Ressource R ON C.Matiere_id = R.Ressource_id;

CREATE VIEW VueNotesGroupes AS
SELECT
    E.groupe_id AS Groupe,
    N.note AS Note,
    C.Controle AS Controle
FROM
    Etudiant E
JOIN
    Notes N ON E.Etudiant_id = N.Etudiant_id
JOIN
    Controle C ON N.Controle_id = C.Controle_id;

CREATE VIEW Statistiquegroupes AS
SELECT
    E.groupe_id AS Groupe,
    AVG(N.note * N.coef) AS Moyenne,
    MAX(N.note) AS Note_Max,
    MIN(N.note) AS Note_Min,
    C.Controle AS Controle
FROM
    Etudiant E
JOIN
    Notes N ON E.Etudiant_id = N.Etudiant_id
JOIN
    Controle C ON N.Controle_id = C.Controle_id
GROUP BY
    E.groupe_id, C.Controle;

CREATE OR REPLACE FUNCTION get_distribution_frequence(controle_id
int)
RETURNS TABLE(intervalle int, plage text, frequence int) AS $$
```

```
BEGIN
    RETURN QUERY
    SELECT
        width_bucket(N.note, min_value, max_value, sac) AS
intervalle,
        '[' || floor(min_value + ((max_value - min_value) /
num_buckets) * (width_bucket - 1)) || ' - ' || floor(min_value +
((max_value - min_value) / num_buckets) * width_bucket) || ']' AS
plage,
        count(*) AS frequence
    FROM
        Notes N
    CROSS JOIN (
        SELECT
            min(note) AS min_value,
            max(note) AS max_value,
            10 AS sac
        FROM
            Notes
        INNER JOIN Controle C ON N.Controle_id = C.Controle_id
        WHERE
            C.Controle_id = controle_id
    ) AS stats
    WHERE
        N.Controle_id = controle_id
    GROUP BY
        intervalle, plage
    ORDER BY
        intervalle;

END;
$$ LANGUAGE plpgsql;
-----------------------------------------------------------------
------------4-------------------------------------------------
CREATE OR REPLACE FUNCTION MesResultats() RETURNS TABLE (
    Matiere varchar(50),
    Controle varchar(50),
    Note decimal(4, 2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT R.Ressource AS Matiere, C.Controle AS Controle, N.note
AS Note
    FROM Notes N
    JOIN Controle C ON N.Controle_id = C.Controle_id
    JOIN Ressource R ON C.Matiere_id = R.Ressource_id
    WHERE N.Etudiant_id = session_user;
END;
```

```
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION saisir_notes_controle(
    ressource_id varchar(10),
    controle_id int,
    controle varchar(50),
    etudiant_id int,
    note decimal(4, 2)
) RETURNS void AS $$
BEGIN
    -- Vérifier si l'enseignant a une entrée existante pour la
même ressource
    IF EXISTS (
        SELECT 1
        FROM Ressource R
        WHERE R.enseignant_id = session_user AND R.Ressource_id =
ressource_id
    ) THEN
        -- Insérer le contrôle s'il n'existe pas déjà
        INSERT INTO Controle (Controle_id, Matiere_id, Controle)
        VALUES (controle_id, ressource_id, controle)
        ON CONFLICT (Controle_id) DO NOTHING;

        -- Insérer la note de l'étudiant pour le contrôle donné
        INSERT INTO Notes (Etudiant_id, Controle_id, note)
        VALUES (etudiant_id, controle_id, note)
        ON CONFLICT (Etudiant_id, Controle_id) DO UPDATE
        SET note = EXCLUDED.note;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION voir_notes_ressources_enseignant()
RETURNS TABLE (
    Etudiant_id int,
    Etudiant_Nom varchar(50),
    Etudiant_Prenom varchar(50),
    Ressource_id varchar(10),
    Ressource varchar(50),
    Controle_id int,
    Controle varchar(50),
    Note decimal(4, 2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        E.Etudiant_id,
        E.Nom AS Etudiant_Nom,
```

```
        E.Prenom AS Etudiant_Prenom,
        R.Ressource_id,
        R.Ressource,
        C.Controle_id,
        C.Controle,
        N.note
    FROM
        Etudiant E
    JOIN
        Notes N ON E.Etudiant_id = N.Etudiant_id
    JOIN
        Controle C ON N.Controle_id = C.Controle_id
    JOIN
        Ressource R ON C.Matiere_id = R.Ressource_id
    WHERE
        R.enseignant_id = session_user;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION modifier_notes_ressource_enseignant(
    etudiant_id int,
    controle_id int,
    note decimal(4, 2)
)
RETURNS void AS $$
BEGIN
    -- Vérifier si l'enseignant a une entrée existante pour la
même ressource
    IF EXISTS (
        SELECT 1
        FROM Ressource R
        JOIN Controle C ON R.Ressource_id = C.Matiere_id
        WHERE R.enseignant_id = session_user AND C.Controle_id =
controle_id
    ) THEN
        -- Mettre à jour la note de l'étudiant pour le contrôle
donné
        UPDATE Notes
        SET note = note
        WHERE Etudiant_id = etudiant_id AND Controle_id =
controle_id;
    END IF;
END;
$$ LANGUAGE plpgsql;
```