

SAE 204 : Exploitation d'une base de données

HAMSEK | Fayçal | Zeus | BUT Info

Table des matières

I-	Mise en place de la base de données.....	2
a.	Cahier des charges.....	2
b.	Schéma Relationnel.....	3
c.	Script de création des tables	4
d.	Procédures stockées.....	6
II-	Visualisation des données	7
a.	Ensembles de données dérivées à visualiser	7
b.	Script des vues pour les données à visualiser	7
III-	Restriction d'accès aux données	8
a.	Définition des règles d'accès aux données :.....	8
b.	Script des règles d'accès aux données :	8
IV-	Conclusion	9

I- Mise en place de la base de données

Cette première partie a pour but de mettre en place notre base de données pour mettre à bien notre projet. Pour cela nous allons faire un cahier des charges à respecter mais aussi un schéma relationnel duquel un script en découlera.

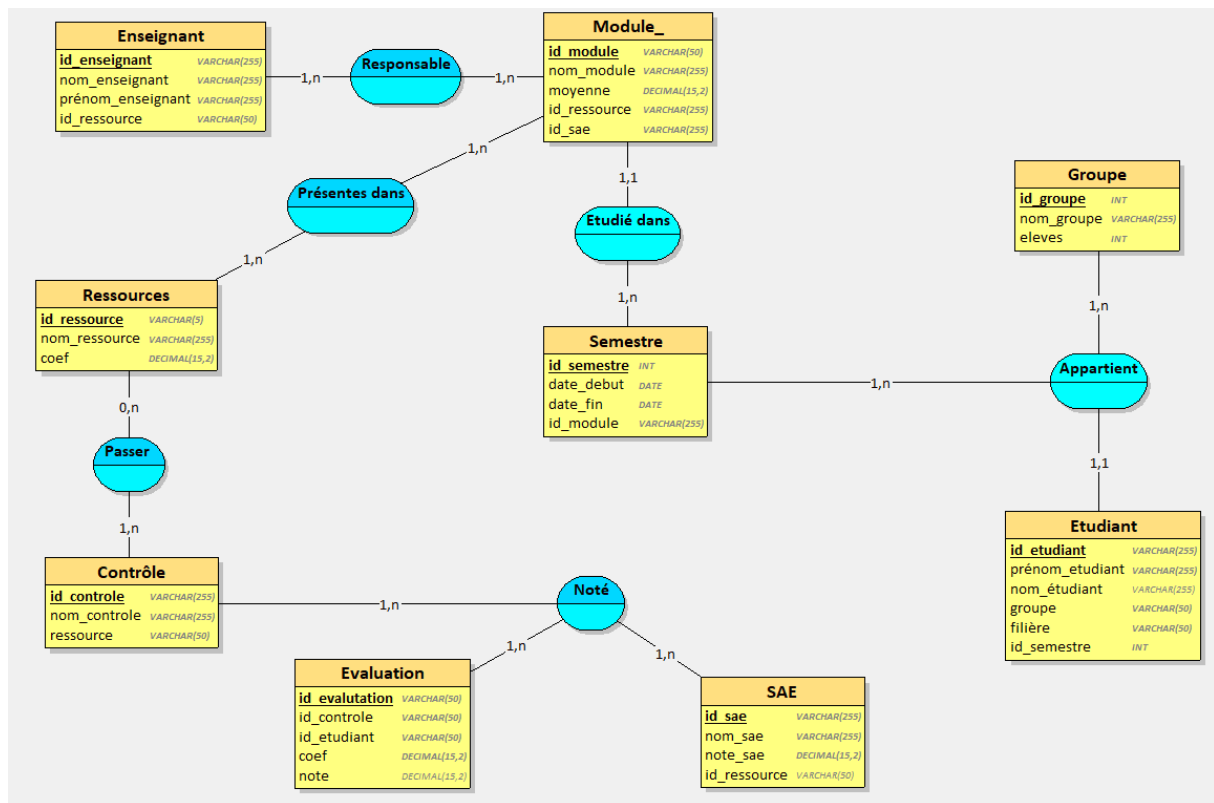
a. Cahier des charges

La base de données doit contenir les tables suivantes :

- Enseignant : elle doit comporter les colonnes suivantes : id (clé primaire), nom, prénom, id_ressource, semestre. La colonne id_ressource est une clé étrangère qui référence la table Ressource.
- Etudiant : elle doit comporter les colonnes suivantes : id (clé primaire), nom, prénom, groupe, filière, id_semestre. La colonne id_semestre est une clé étrangère qui référence la table Semestre.
- Module : elle doit comporter les colonnes suivantes : id_module (clé primaire), nom_module, moyenne, id_ressource, id_sae. Les colonnes id_ressource et id_sae sont des clés étrangères qui référencent les tables Ressource et SAE respectivement.
- Evaluation : elle doit comporter les colonnes suivantes : id_evaluation (clé primaire), id_controle, id_etudiant, coef, note.
- SAE : elle doit comporter les colonnes suivantes : id_sae (clé primaire), nom_SAE, id_ressource, note. La colonne id_ressource est une clé étrangère qui référence la table Module.
- Semestre : elle doit comporter les colonnes suivantes : id_semestre (clé primaire), date_debut, date_fin, id_module. La colonne id_module est une clé étrangère qui référence la table Ressource.
- Groupe : elle doit comporter les colonnes suivantes : id_groupe (clé primaire), nom_groupe, eleves.
- Ressource : elle doit comporter les colonnes suivantes : id_ressource (clé primaire), nom_ressource, Coef.

Les clés étrangères doivent être correctement définies pour assurer l'intégrité de la base de données.

b. Schéma Relationnel



c. Script de création des tables

```
-- Création de la table Ressource
CREATE TABLE Ressource (
    id_ressource VARCHAR(200) PRIMARY KEY,
    nom_ressource VARCHAR(200),
    Coef DECIMAL
);

-- Création de la table SAE
CREATE TABLE SAE (
    id_sae VARCHAR(200) PRIMARY KEY,
    nom_SAE VARCHAR(200),
    id_ressource VARCHAR(200),
    note DOUBLE PRECISION,
    FOREIGN KEY (id_ressource) REFERENCES Ressource(id_ressource)
);

-- Création de la table Module
CREATE TABLE Module1 (
    id_module VARCHAR(200) PRIMARY KEY,
    nom_module VARCHAR(200),
    moyenne DECIMAL,
    id_ressource VARCHAR(200),
    id_sae VARCHAR(200),
    FOREIGN KEY (id_ressource) REFERENCES Ressource(id_ressource),
    FOREIGN KEY (id_sae) REFERENCES SAE(id_sae)
);

-- Création de la table Semestre
CREATE TABLE Semestre (
    id_semestre INT PRIMARY KEY,
    date_debut DATE,
    date_fin DATE,
    id_module VARCHAR(200),
    FOREIGN KEY (id_module) REFERENCES Module1(id_module)
);

-- Création de la table Groupe
CREATE TABLE Groupe (
    id_groupe INT PRIMARY KEY,
    nom_groupe VARCHAR(200),
    eleves INT
);

-- Création de la table Enseignant
```

```

CREATE TABLE Enseignant (
    id_enseignant SERIAL PRIMARY KEY,
    nom_enseignant VARCHAR(200),
    prénom_enseignant VARCHAR(200),
    id_ressource VARCHAR(200),
    semestre INT,
    FOREIGN KEY (id_ressource) REFERENCES Ressource(id_ressource)
);

-- Création de la table Etudiant
CREATE TABLE Etudiant (
    id_etudiant SERIAL PRIMARY KEY,
    nom_etudiant VARCHAR(200),
    prénom_etudiant VARCHAR(200),
    groupe VARCHAR(200),
    filière VARCHAR(200),
    id_semestre INT,
    FOREIGN KEY (id_semestre) REFERENCES Semestre(id_semestre)
);

-- Création de la table Controle
CREATE TABLE Controle (
    id_controle VARCHAR(200) PRIMARY KEY,
    id_module VARCHAR(200),
    date_controle DATE,
    FOREIGN KEY (id_module) REFERENCES Module1(id_module)
);

-- Création de la table Evaluation
CREATE TABLE Evaluation (
    id_evaluation VARCHAR(200) PRIMARY KEY,
    id_controle VARCHAR(200),
    id_etudiant VARCHAR(200),
    coef DECIMAL,
    note DECIMAL
);

```

d. Procédures stockées

```
-- retirer une note
CREATE OR REPLACE PROCEDURE retirer_note(
    p_id_evaluation VARCHAR
)
AS $$
BEGIN
    DELETE FROM Evaluation
    WHERE id_evaluation = p_id_evaluation;
END;
$$ LANGUAGE plpgsql;

--ajouter une SAE
CREATE OR REPLACE PROCEDURE ajouter_sae(
    p_id_sae VARCHAR,
    p_nom_sae VARCHAR,
    p_id_ressource VARCHAR,
    p_note DOUBLE PRECISION
)
AS $$
BEGIN
    INSERT INTO SAE (id_sae, nom_sae, id_ressource, note)
    VALUES (p_id_sae, p_nom_sae, p_id_ressource, p_note);
END;
$$ LANGUAGE plpgsql;

-- ajouter une note a un groupe
CREATE OR REPLACE FUNCTION ajouter_note_groupe(
    id_groupe INT,
    id_evaluation VARCHAR(200),
    id_controle VARCHAR(200),
    id_etudiant VARCHAR(200),
    coef DECIMAL,
    note DECIMAL
)
RETURNS VOID AS $$
BEGIN
    INSERT INTO Evaluation(id_evaluation, id_controle, id_etudiant, coef,
note)
    VALUES(id_evaluation, id_controle, id_etudiant, coef, note);

    UPDATE Etudiant
    SET moyenne = (
        SELECT AVG(note * coef)
        FROM Evaluation
```

```

        WHERE Evaluation.id_etudiant = Etudiant.id_etudiant
    )
    WHERE Etudiant.id_groupe = id_groupe;
END;
$$ LANGUAGE plpgsql;

```

II- Visualisation des données

Dans cette deuxième partie nous allons faire des fonctions pour voir certaines données de notre base de données et nous allons donc utiliser VIEW et des procédures stockées.

a. Ensembles de données dérivées à visualiser

- L'ensemble des professeurs
- Le relevé de note du groupe G1
- L'ensemble des élèves d'un groupe

b. Script des vues pour les données à visualiser

```

--voir la liste des enseignants
CREATE VIEW Enseignants_View AS
SELECT id_enseignant, nom_enseignant, prénom_enseignant
FROM Enseignant;

-- voir les notes du groupe G1
CREATE VIEW Notes_Groupe_View AS
SELECT Etudiant.nom_etudiant, Etudiant.prénom_etudiant, Evaluation.note
FROM Etudiant
JOIN Evaluation ON CAST(Etudiant.id_etudiant AS VARCHAR) = Evaluation.id_etu-
diant
WHERE Etudiant.groupe = 'G1';

-- voir les etudiants des groupes
CREATE VIEW Etudiant_Groupe_View AS
SELECT Etudiant.id_etudiant, Etudiant.nom_etudiant, Etudiant.prénom_etudiant,
Groupe.nom_groupe, Groupe.filière
FROM Etudiant
INNER JOIN Groupe ON Etudiant.groupe = Groupe.nom_groupe;

```

III- Restriction d'accès aux données

Dans cette troisième et dernière partie nous allons fixer des règles à notre base de données car tout le monde ne peut pas la modifier et donc éviter des soucis de gestion. Nous allons faire différentes fonctions pour cela.

a. Définition des règles d'accès aux données :

- Un enseignant ne peut consulter que les notes du module dont il est responsable
- Un enseignant peut supprimer une note d'un module dont il est responsable
- Un étudiant ne peut pas modifier les notes

b. Script des règles d'accès aux données :

```
--un enseignant peut supprimer une note d'un module dont il est responsable
CREATE OR REPLACE FUNCTION supprimer_note_responsable(p_enseignant_id INTEGER,
p_module_id VARCHAR(200), p_etudiant_id VARCHAR(200), p_evaluation_id VARCHAR(200))
RETURNS VOID
AS $$
BEGIN
    DELETE FROM Evaluation e
    USING Controle c
    INNER JOIN Module m ON c.id_module = m.id_module
    INNER JOIN Ressource r ON m.id_ressource = r.id_ressource
    WHERE e.id_evaluation = p_evaluation_id
        AND e.id_etudiant = p_etudiant_id
        AND c.id_module = p_module_id
        AND r.id_ressource = ANY (
            SELECT id_ressource
            FROM Enseignant
            WHERE id_enseignant = p_enseignant_id
        );
END;
$$ LANGUAGE plpgsql;

-- une étudiant ne peut pas modifier les notes
CREATE FUNCTION update_evaluation_forbidden() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        RAISE EXCEPTION 'Vous n'êtes pas autorisé à modifier les notes';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



```

-- il lui faut son trigger
CREATE TRIGGER evaluation_update_forbidden
BEFORE UPDATE ON Evaluation
FOR EACH ROW
EXECUTE FUNCTION update_evaluation_forbidden();
-- un enseignant ne peut consulter que les notes de la ressource dont il est
responsable
CREATE FUNCTION get_notes_responsable(p_enseignant_id INTEGER)
RETURNS TABLE (id_evaluation VARCHAR(200), id_controle VARCHAR(200), id_etu-
diant VARCHAR(200), coef DECIMAL, note DECIMAL)
AS $$
BEGIN
    RETURN QUERY
    SELECT e.id_evaluation, e.id_controle, e.id_etudiant, e.coef, e.note
    FROM Evaluation e
    INNER JOIN Controle c ON e.id_controle = c.id_controle
    INNER JOIN Module m ON c.id_module = m.id_module
    INNER JOIN Ressource r ON m.id_ressource = r.id_ressource
    WHERE r.id_ressource = ANY (
        SELECT id_ressource
        FROM Enseignant
        WHERE id = p_enseignant_id
    );
END;
$$ LANGUAGE plpgsql;

```

IV- Conclusion

Durant cette SAE j'ai vu de nombreux éléments que j'avais déjà utilisé sur la création et l'exploitation de base de données, les schémas relationnels, la réalisation de script. Mais j'ai pu m'initier à de nouveaux éléments et les mettre en place comme les procédures stockées, les vues et les règles de gestions. Ceci m'a permis d'acquérir des compétences mais d'en renforcer d'autres.