1 ère année BUT INFO
L'Iut de Villetaneuse, Villetaneuse

Base de Données et langage SQL

Rapport de la SAE1.04

Réalisé par : Hiba Oumsid

Enseignant : M. Laurent Audibert

Rendre le : 15 janvier

Introduction

Dans le cadre de notre première année en BUT Info, nous avons vu comment créer des tables de base de données grâce à des requêtes SQL.

Cette SAE a pour objectif d'approfondir notre connaissance en base de données et en langages SQL à travers **Power Architect** (AGL) et **PostgreSQL.**

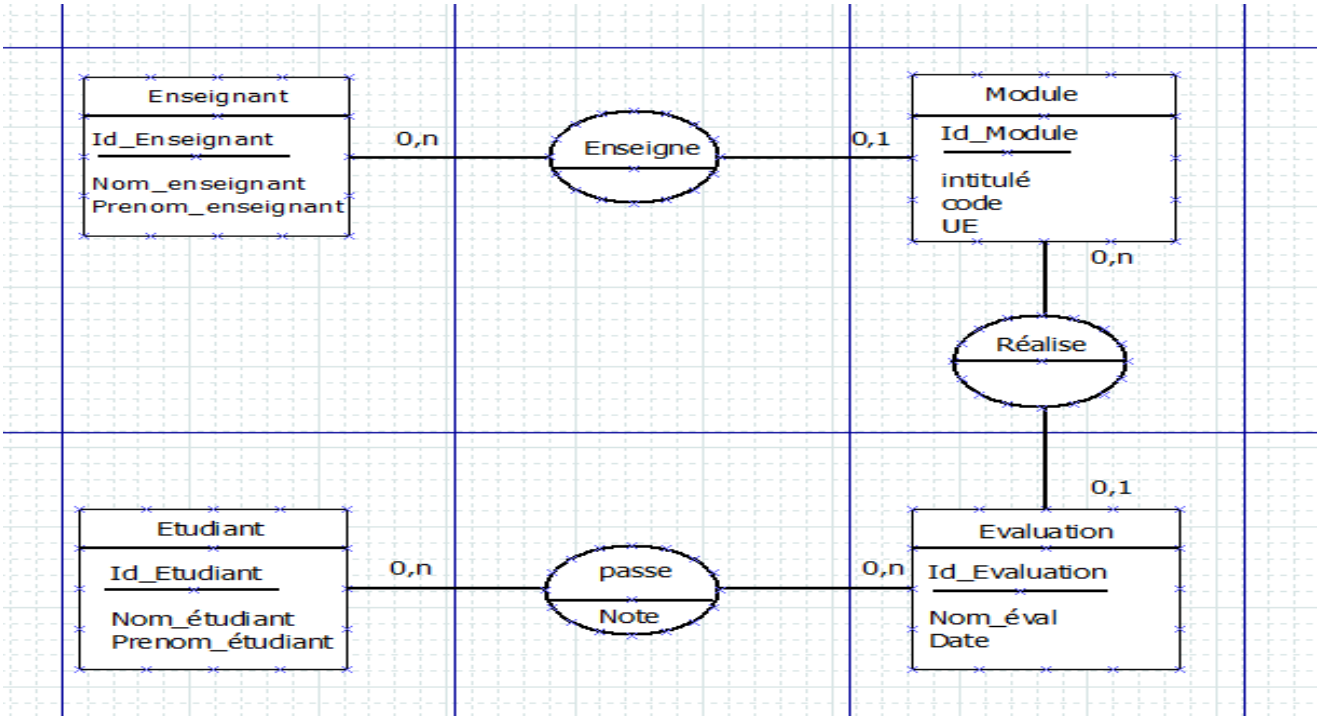
La situation de modélisation est la suivante :

Pour un module, on devra connaitre son intitulé, son code, son unité d'enseignement et son responsable. Le **responsable d'un module** peut réaliser autant d'évaluations qu'il veut. Chaque évaluation doit être mémorisée. **Une évaluation** comporte un nom et une date. Bien entendu, il faut stocker les **notes des étudiants** à chacune des évaluations. On veut également conserver le nom et le prénom **des enseignants** et **des étudiants** et être capable de les distinguer.

Sommaire

A. Modélisation et script de création « sans AGL »

1) Modèle entités-associations avec DIA



2) Schéma relationnel

Le Schéma relationnel est le suivant **:**

Module (Id_Module, code, UE, intitulé, Id_Enseignant) où Id_Enseignant fait référence à Enseignant.

Enseignant (Id_enseignant, Nom_enseignant, prénom_enseignant)

Etudiant (Id_étudiant, Nom_étudiant, Prénom_étudiant)

Evaluation (Id_Evaluation, Nom_éval, Date, Id_Module) où Id_Module fait référence à Module.

Passe (Id_Etudiant, Id_Evaluation, Note) où Id_Etudiant fait référence à Etudiant et Id_Evaluation fait référence à Evaluation.

3) Script SQL de création des tables

```
CREATE TABLE Enseignant (
        Id_Enseignant INT PRIMARY KEY,
        Nom_Enseignant VARCHAR NOT NULL,
        Preneom_Enseignant VARCHAR NOT NULL
) ;

CREATE TABLE Module (
        id_Module INTEGER PRIMARY KEY,
        intitule VARCHAR NOT NULL,
        Code VARCHAR NOT NULL,
        UE VARCHAR NOT NULL,
        Id_Enseignant REFERENCES Enseignant (Id_Enseignant)
) ;

CREATE TABLE Evaluation (
        Id_Evaluation INTEGER PRIMARY KEY NOT NULL,
         date DATE NOT NULL,
         Nom_Evaluation VARCHAR NOT NULL,
        Id_Module INTEGER REFERENCES Module (Id_Module)
) ;

CREATE TABLE Etudiant (
        Id_Etudiant INTEGER PRIMARY KEY NOT NULL,
         Nom_Etudiant VARCHAR NOT NULL,
        Prenom_Etudiant VARCHAR NOT NULL
) ;


CREATE TABLE Passe (
        Id_Etudiant INTEGER REFERENCES Etudiant (Id_Etudiant) ON DELETE
        CASCADE,
        Id_Evaluation INTEGER REFERENCES Evaluation (Id_Evaluation) ON DELETE
        CASCADE,
        PRIMARY KEY (Id_Evaluation, Id_Etudiant),
        Note REAL
);
```

B. Modélisation et script de création « avec AGL »

1) Illustration d'une association fonctionnelle

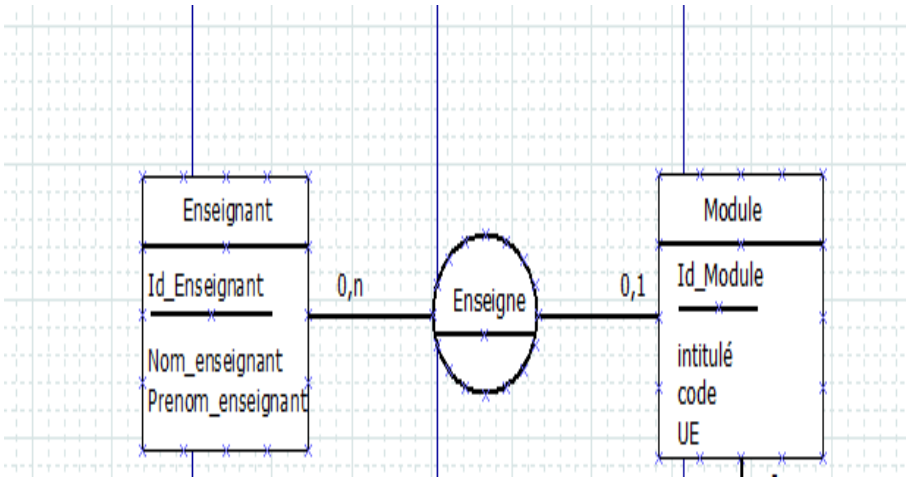


Figure1 : sans AGL

Figure 2 : Avec AGL

2) Illustration d'une association maillée

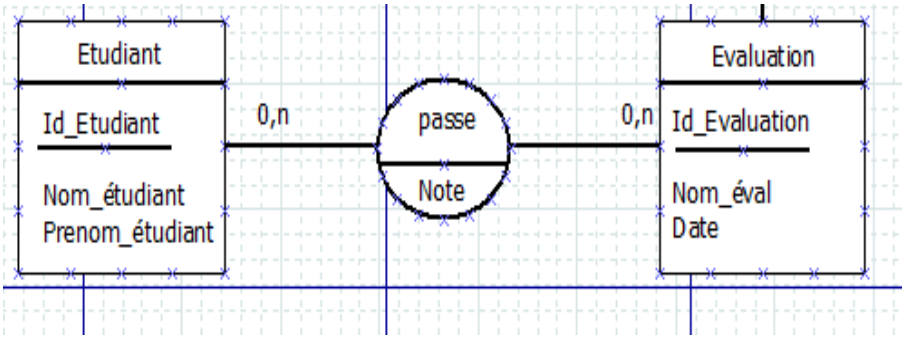


Figure : Sans AGL

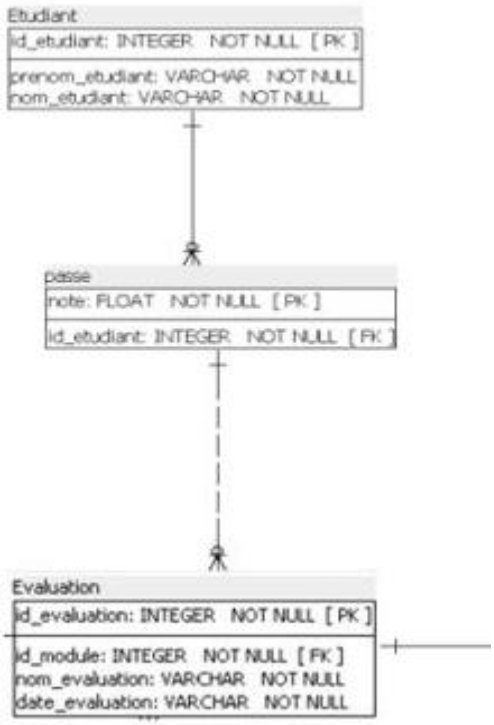
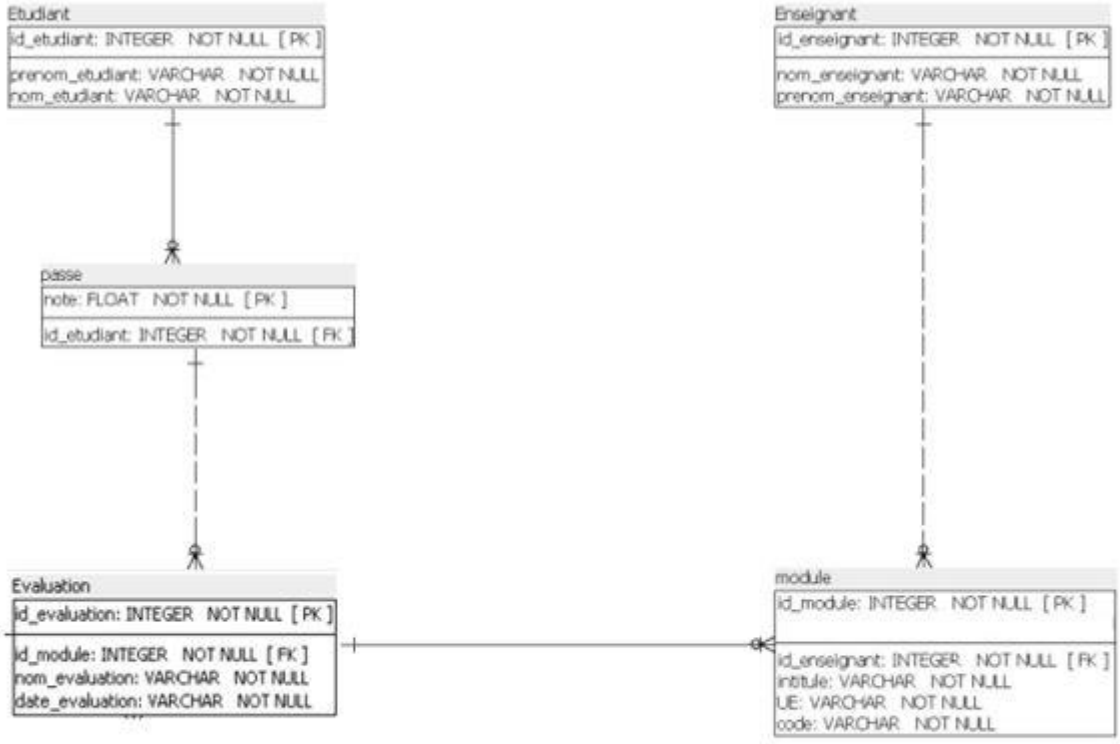


Figure2 : avec Power Architect

3) Modèle entités-associations avec Power Architect



⇨ Concernant **les points communs** entre les schémas entités-associations faites à la main et sur Power Architect :

On remarque que les clés étrangères sont les premiers attributs des tables,

⇨ Concernant **les différents points** entre les schémas faites sur AGL et à la main

Sur l'AGL : à côté de la clé primaire est écrit [PK] qui fait référence à PRIMARY KEY, on remarque que à côté de La clé étrangère est écrit [FK] comme FOREIGN KEY, la clé étrangère se met à la fin de la création des entités afin de mettre la relation entre les tables, la relation entre les tables est exprimée sous fourme de flèche et à la fin on remarque l'absence des cardinalités des relations.

Sur **le schéma fait à la main** : la clé primaire est soulignée et à coté de chaque table est indiqué la cardinalité des relations,

4) Script SQL généré par l'AGL

Ci-dessous le script qui a été généré par Power Architect.

```
CREATE TABLE Enseignant (
        Id_Enseignant INTEGER NOT NULL,
        Nom_Enseignant VARCHAR NOT NULL,
        Prenom_Enseignant VARCHAR NOT NULL,
        CONSTRAINT Enseignant_pk PRIMARY KEY (Id_Enseignant)
);


CREATE TABLE Etudiant (
        Id_Etudiant INTEGER NOT NULL,
        Prenom_Etudiant VARCHAR NOT NULL,
        Nom_Etudiant VARCHAR NOT NULL,
        CONSTRAINT Id_Etudiant PRIMARY KEY (Id_Etudiant)
);


CREATE TABLE passe (
        note FLOAT NOT NULL,
        Id_Etudiant INTEGER NOT NULL,
        CONSTRAINT passe_pk PRIMARY KEY (note)
);
```

```
CREATE TABLE evaluation (
        Id_Evaluation INTEGER NOT NULL,
        date DATE NOT NULL,

         Nom_Evaluation VARCHAR NOT NULL,

        note REAL NOT NULL,
        CONSTRAINT Evaluation_pk PRIMARY KEY (Id_Evaluation)
);

CREATE TABLE module (
        Id_Module INTEGER NOT NULL,
        Id_Evaluation INTEGER NOT NULL,
        Id_Enseignant INTEGER NOT NULL,
        intitule VARCHAR NOT NULL,
        UE VARCHAR NOT NULL,
        code VARCHAR NOT NULL,
        CONSTRAINT Module_pk PRIMARY KEY (Id_Module, Id_Evaluation)
);

ALTER TABLE module ADD CONSTRAINT Enseignant_module_fk
FOREIGN KEY (Id_Enseignant)
REFERENCES Enseignant (Id_Enseignant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE passe ADD CONSTRAINT Etudiant_passe_fk
FOREIGN KEY (Id_Etudiant)
REFERENCES Etudiant (Id_Etudiant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;


ALTER TABLE evaluation ADD CONSTRAINT passe_evaluation_fk
FOREIGN KEY (note)
REFERENCES passe (note)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

```
ALTER TABLE module ADD CONSTRAINT Evaluation_module_fk
FOREIGN KEY (Id_Evaluation)
REFERENCES evaluation (Id_Evaluation)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE
```

5) Comparison des deux scripts

On remarque que Le script qui concerne la création des tables généré par l'AGL est presque le même script fait manuellement.
On trouve dans le script de Power Architect une structure ALTER TABLE, cette requête permet d'apporter des modifications sur les tables comme ajouter des colonnes, ou les supprimer ou même ajouter des contraintes avec la structure ADD CONSTRAINT.
Ainsi qu'on constate l'apparition de deux structures à la fin du script comme « ON UPDATE NO ACTION » qui sert à gérer les modifications sur la colonne référencée par une clé étrangère ; si la colonne référencée est modifiée, la base de données ne mettra pas les modifications automatiquement dans la colonne où se situe la clé étrangère.

C. Peuplement des tables et requêtes

1) Script de peuplement

- ⇨ La structure de la requête est la suivante : \COPY TABLE (col1, col2…cloln) FROM 'chemin du fichier' DELEMITER 'délimiteur' FORMAT OPTION ;
- ⇨ Dans OPTION j'ai utilisé HEADER, afin de ne pas prendre en compte la 1 ère ligne du fichier.

Le script SQL est le suivant :

```
\COPY Enseignant (Id_Enseignant, Nom_Enseignant, Prenom_Enseignant) FROM
    ' C:\Users\Hiba\data.csv' DELIMITER ';' CSV HEADER;
\COPY Module (Id_Module, code, UE, Intitule, Id_Enseignant) FROM
    'C:\Users\Hiba\data.csv' DELIMITER ';' CSV HEADER;
```

```
\COPY Evaluation (Id_Evaluation, Nom_Evaluation, date, Id_Module) FROM
        'C:\Users\Hiba\data.csv' DELIMITER ';' CSV HEADER;

\COPY Etudiant (Id_Etudiant, Nom_Etudiant, Prenom_Etudiant, Id_Evaluation)
        FROM 'C:\Users\Hiba\data.csv' DELIMITER ';' CSV HEADER;
```

2) Présentation de deux requêtes SQL sur la base de données

```
SELECT Prenom_Etudiant//' '//Nom_Etudiant//'a eu la note de' //note//'dans l'évaluation'//Nom_Evaluation FROM Etudiant JOIN Passer ON Id_Etudiant=Id_Etudiant, Id_Evaluation JOIN Evaluation USING (Id_Evaluation) ;
```

⇨ Cette requête permet d'avoir comme résultat des phrases où on trouvera dans chaque phrase le nom et le prénom de l'étudiant ainsi que la note qu'il a eu dans une matière en précisant le nom de l'évaluation

```
SELECT Nom_Etudiant, Prenom_Etudiant, COUNT (*) AS Nbr_d_Etudiants,
    MIN(Note) AS max_promo,
    MAX(Note) AS min_promo FROM Etudiant JOIN Passer ON
    Id_Etudiant=Id_Etudiant, Id_Evaluation JOIN Evaluation USING (Id_Evaluation)
    GROUP BY Nom_Etudiant, Prenom_Etudiant
    ORDER BY Nom_Etudiant;
```

⇨ Cette requête nous permettra de sélectionner les noms, les prénoms et les notes des étudiants ainsi que la liste va être ordonnée selon l'ordre alphabétiques des noms