Rapport SAE S104

Création d'une base de données : notations

Bases de Données et langage SQL

But informatique S1 – TLALOC

Camélia ANTOINE

Université Sorbonne Paris Nord - IUT de Villetaneuse

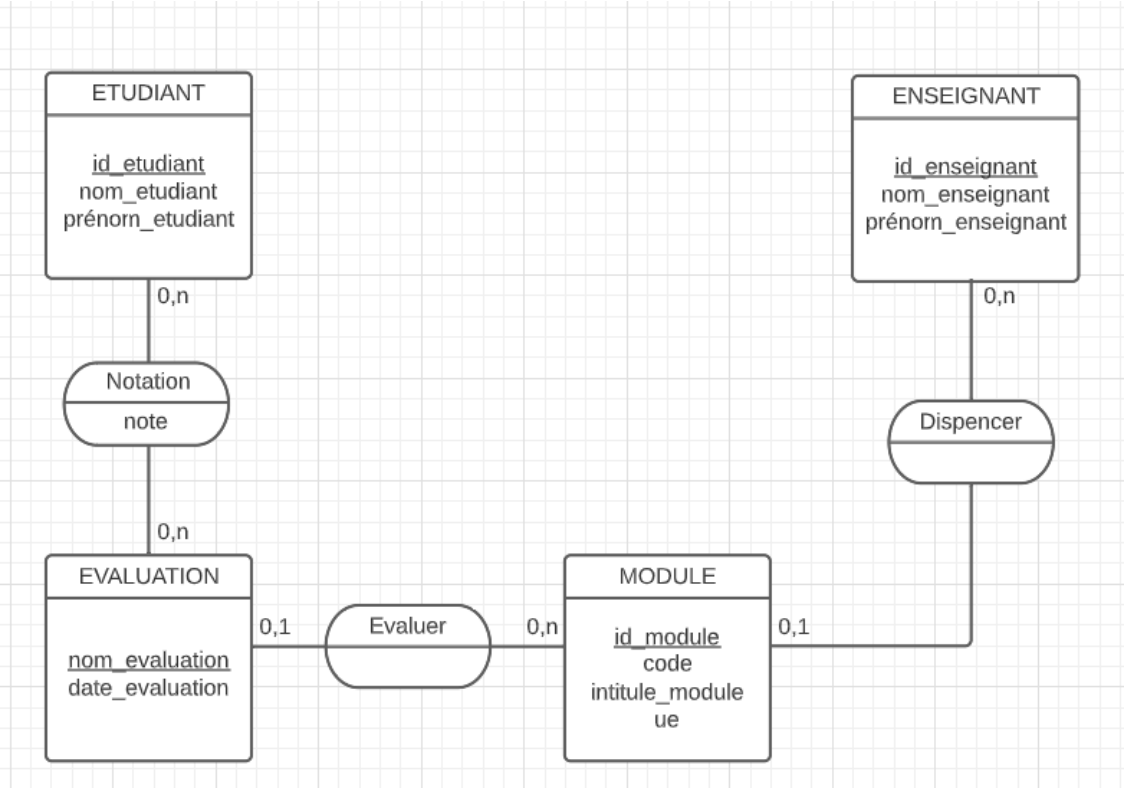
Enseignante : M. Vallée

Date : 15 Janvier 2023

SOMMAIRE

1. Modélisation et script de création “sans AGL”

1. Modèle entités-associations



2. Schéma relationnel

Légende : * correspond à une clé étrangère

ETUDIANT(id_etudiant, nom_etudiant, prenom_etudiant)

ENSEIGNANT(id_enseignant, nom_enseignant, prenom_etudiant)

EVALUATION(nom_evaluation, id_module*, date_evaluation, id_etudiant*)

MODULE(id_module, code, id_enseignant*, intitule_module, ue)

NOTATION(id_etudiant*, nom_evaluation*, note)

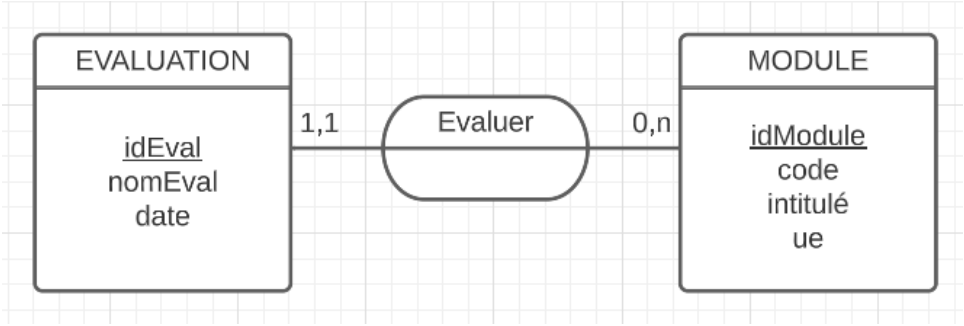
3. Script SQL de création des tables

```
CREATE TABLE Etudiant (
      id_etudiant INTEGER PRIMARY KEY,
      nom_etudiant VARCHAR NOT NULL,
      prenom_etudiant VARCHAR NOT NULL);

CREATE TABLE Enseignant (
      id_enseignant INTEGER PRIMARY KEY,
      nom_enseignant VARCHAR NOT NULL,
      prenom_enseignant VARCHAR NOT NULL);

CREATE TABLE Module (
      id_module INTEGER PRIMARY KEY,
      code VARCHAR NOT NULL,
      intitule_module VARCHAR NOT NULL,
      ue VARCHAR NOT NULL,
      id_enseignant INTEGER REFERENCES Enseignant ON DELETE CASCADE);

CREATE TABLE Evaluation (
      nom_evaluation VARCHAR PRIMARY KEY,
      date_evaluation DATE NOT NULL,
      id_module INTEGER REFERENCES Module ON DELETE CASCADE,
      id_etudiant INTEGER REFERENCES Etudiant ON DELETE CASCADE);

CREATE TABLE Notation (
      id_etudiant INTEGER REFERENCES Etudiant ON DELETE CASCADE,
      nom_evaluation VARCHAR REFERENCES Evaluation ON DELETE CASCADE,
      note FLOAT NOT NULL,
      PRIMARY KEY (id_etudiant, nom_evaluation));
```

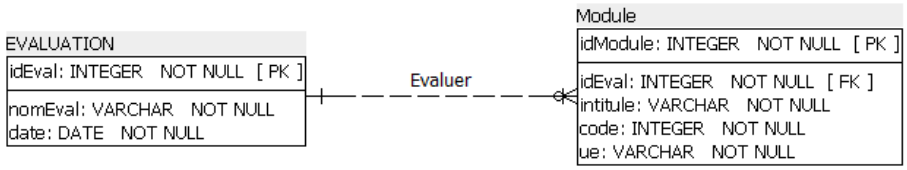
2. Modélisation et script de création “avec AGL”

1.Illustrations comparatives cours/AGL d’une association fonctionnelle

Association fonctionnelle (cours) :



Association fonctionnelle (AGL) :



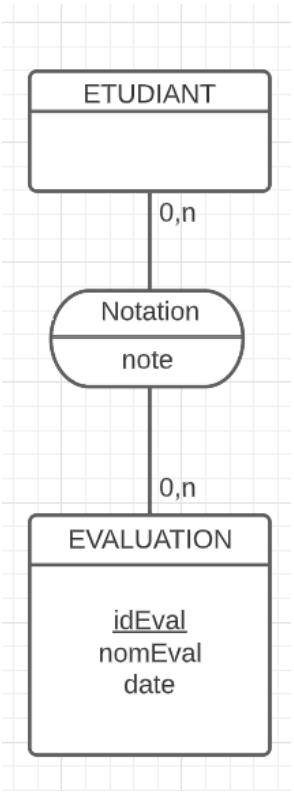
Commentaire :

Dans l’association du cours le type association est représenté par un “cercle” et ses cardinalités sous forme de chiffre. Dans les types entités il y a que la clé primaire soulignées et il y a les attributs. Les clés étrangères sont sous-entendues grâce aux cardinalités et n’apparaissent pas.

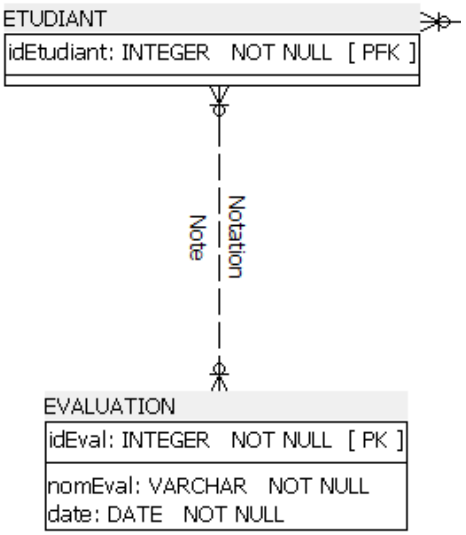
Dans l’association AGL le type association est représenté par un lien (non identifié) et les cardinalités sont représentées sous forme de symboles. Dans les types entités il y a que la clé primaire dans une ligne précisée comme ceci : [PK] soit Primary Key. Il y a les attributs dans la ligne en dessous mais aussi la clé étrangère précisée [FK] soit Foreign Key. De plus, le domaine des valeurs de chaque clé est précisé (INTEGER, VARCHAR, …). Pour finir, il est précisé NOT NULL donc les valeurs de ces attributs sont obligatoires.

2. Illustrations comparatives cours/AGL d'une association maillée

Association maillée (cours) :



Association maillée (AGL) :

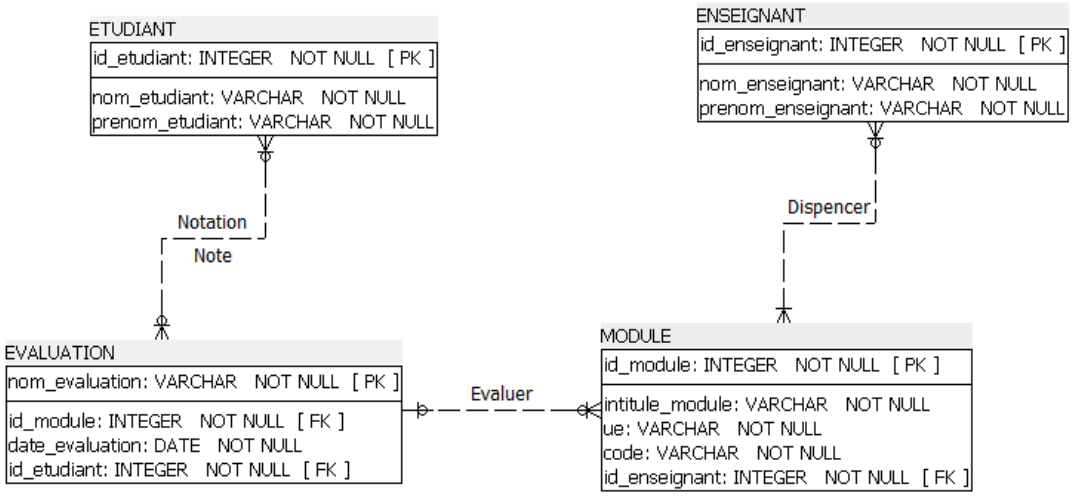


Commentaire :

Dans l'association du cours le type association a un attribut et est représenté par un "cercle" ainsi que ses cardinalités sous forme de chiffre. Dans les types entités il y a que la clé primaire soulignée et il y a les attributs. Les clés étrangères sont sous-entendues grâce aux cardinalités et n'apparaissent pas. Le type entité ETUDIANT n'a aucune clé ou attribut car il y a une spécialisation avec PERSONNE (qu'on ne voit pas ici).

Dans l'association AGL le type association est représenté par un lien (non identifié) et les cardinalités sont représenté sous forme de symboles. Dans les types entités il y a que la clé primaire dans une ligne précisée comme ceci : [PK] soit Primary Key. Cette fois ETUDIANT à pour clé primaire étrangère précisée [PFK] soit Primary Foreign Key. Cette clé vient de la table PERSONNE non visible ici. De plus, le domaine des valeurs de chaque clé est précisé (INTEGER, VARCHAR, …). Pour finir, il est précisé NOT NULL donc les valeurs de ces attributs sont obligatoires.

3.Modèle entités-associations réalisé avec AGL



4.Script SQL de création des tables généré automatiquement par AGL

```
CREATE TABLE public.ENSEIGNANT (
                id_enseignant INTEGER NOT NULL,
                nom_enseignant VARCHAR NOT NULL,
                prenom_enseignant VARCHAR NOT NULL,
                CONSTRAINT id_enseignant PRIMARY KEY (id_enseignant)
);


CREATE TABLE public.MODULE (
                id_module INTEGER NOT NULL,
                intitule_module VARCHAR NOT NULL,
                ue VARCHAR NOT NULL,
                code VARCHAR NOT NULL,
                id_enseignant INTEGER NOT NULL,
                CONSTRAINT id_module PRIMARY KEY (id_module)
```

```
);


CREATE TABLE public.ETUDIANT (
                id_etudiant INTEGER NOT NULL,
                nom_etudiant VARCHAR NOT NULL,
                prenom_etudiant VARCHAR NOT NULL,
                CONSTRAINT id_etudiant PRIMARY KEY (id_etudiant)
);


CREATE TABLE public.EVALUATION (
                nom_evaluation VARCHAR NOT NULL,
                id_module INTEGER NOT NULL,
                date_evaluation DATE NOT NULL,
                id_etudiant INTEGER NOT NULL,
                CONSTRAINT      nom_evaluation      PRIMARY      KEY
          (nom_evaluation)
);


ALTER TABLE public.MODULE ADD CONSTRAINT enseignant_module_fk
FOREIGN KEY (id_enseignant)
REFERENCES public.ENSEIGNANT (id_enseignant)
ON DELETE CASCADE
ON UPDATE CASCADE
NOT DEFERRABLE;

ALTER TABLE public.EVALUATION ADD CONSTRAINT module_evaluation_fk
FOREIGN KEY (id_module)
```

```
REFERENCES public.MODULE (id_module)
ON DELETE CASCADE
ON UPDATE CASCADE
NOT DEFERRABLE;

ALTER TABLE public.EVALUATION ADD CONSTRAINT etudiant_evaluation_fk
FOREIGN KEY (id_etudiant)
REFERENCES public.ETUDIANT (id_etudiant)
ON DELETE CASCADE
ON UPDATE CASCADE
NOT DEFERRABLE;
```

5.Différences entre les scripts produit manuellement et automatiquement

Dans le script généré automatiquement la table Notation n'apparaît pas car elle reste un simple type association. Ceci s'explique par le fait que le script se génère par rapport au modèle entité association créé.

Dans mon code les clés étrangères sont créées grâce à : `INTEGER REFERENCES` table (d'où elle vient) et ses contraintes (`ON DELETE CASCADE` par exemple) alors que dans le script généré, des tables `ALTER` sont créés où la clé étrangère est précisée `FOREIGN KEY` ainsi que de qu'elle table elle vient. Il y a aussi toutes ses contraintes.

De plus, dans mon code une clé est directement définie `PRIMARY KEY` lors de sa création alors que dans le script généré la clé est créée et à la fin on précise que c'est la clé primaire : `CONSTRAINT cle PRIMARY KEY (cle)`.

Pour finir, en plus de la syntaxe, la forme change. L'indentation du script généré est beaucoup plus grande.

3. Peuplement des tables et requêtes

1.Description commentée des différentes étapes de votre script de peuplement

Il faut d'abord créer une table temporaire dont les colonnes sont exactement les mêmes que celle du fichier data.csv afin d'y mettre tous les éléments :

```
CREATE TABLE Temporaire (id_enseignant INTEGER, nom_enseignant VARCHAR, prenom_enseignant VARCHAR, id_module INTEGER, code VARCHAR, ue VARCHAR, intitule_module VARCHAR, nom_evaluation VARCHAR, date_evaluation DATE, note FLOAT, id_etudiant INTEGER, nom_etudiant VARCHAR, prenom_etudiant VARCHAR);
```

```
COPY Temporaire FROM 'C:\Users\Public\data.csv' DElIMITER';'CSV HEADER;
```

Ensuite depuis cette table il faut insérer les valeurs dans chaque table correspondante :

```
INSERT INTO Etudiant (SELECT DISTINCT id_etudiant, nom_etudiant, prenom_etudiant FROM Temporaire);

INSERT INTO Enseignant (SELECT DISTINCT id_enseignant, nom_enseignant, prenom_enseignant FROM Temporaire);

INSERT INTO Module (SELECT DISTINCT id_module, code, intitule_module, ue, id_enseignant FROM Temporaire);

INSERT INTO Evaluation (SELECT DISTINCT nom_evaluation, date_evaluation, id_module, id_etudiant FROM Temporaire);

INSERT INTO Notation (SELECT DISTINCT id_etudiant, nom_evaluation, note FROM Temporaire);
```

2.Présentation commentée de deux requêtes intéressantes sur la base de données

Requête 1 :

```
SELECT nom_etudiant, prenom_etudiant, nom_evaluation, note
FROM Etudiant JOIN Notation USING (id_etudiant)
WHERE Notation.nom_evaluation = 'Contrôle 1' AND Notation.note<10
ORDER BY note ASC;
```

En quoi cette requête est intéressante :

Ici, ce qui est intéressant, c'est le résultat. En effectuant cette requête on obtient un tableau des étudiant ayant fait le 'contrôle 1' et aillant eu moins de 10 (soit la moyenne) dans l'ordre de la note la plus basse à la plus élevée. Grâce à une telle requête on peut observer qui des élèves ayant passés ce contrôle ont besoin d'aide, du cas le plus urgent au moins urgent.

Requête 2 :

```
SELECT code, intitule_module, nom_evaluation, AVG(note) as Moyenne_des_notes
FROM Module JOIN Evaluation USING (id_module) JOIN Notation USING (nom_evaluation)
GROUP BY code, intitule_module, nom_evaluation
HAVING AVG(note)<10
ORDER BY AVG(note);
```

En quoi cette requête est intéressante :

Ici, c'est la requête qui est intéressante. On sélectionne le code et l'intitulé du module, le nom de l'évaluation et la moyenne (avec un alias) obtenue par la promo à cette évaluation. On regroupe l'évaluation et le module (son intitulé et son code). On garde seulement les moyennes en dessous de 10 ce qui dans le monde réel pourrait montrer les matières problématiques. Enfin, on ordonne dans l'ordre croissant les moyennes. Cette requête utilise plusieurs commandes pour cibler un résultat précis ce qui la rend intéressante.