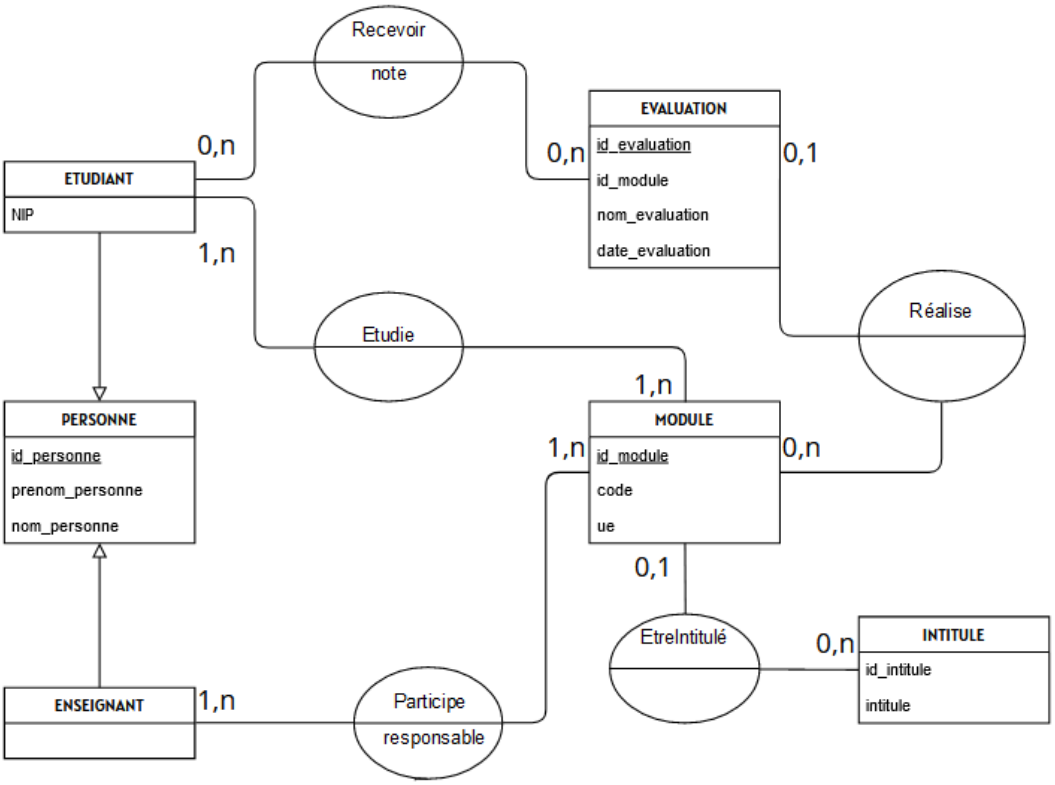
SAE BASE DE DONNEES :

SAE 104 : Création d'une base de données faite par India CABO du Groupe Shango en BUT1 Informatique.

SOMMAIRE :

2.1 : Modélisation et script de création “sans AGL”

1. Modèle entités-associations respectant la syntaxe de cours



2. Schéma relationnel

PERSONNE (id_personne, nom_personne, prenom_personne)

Ce schéma de relation rassemble les informations concernant les personnes.

ETUDIANT (id_etudiant, NIP)

Ce schéma de relation rassemble les informations concernant les étudiants.

id_etudiant est une clé étrangère faisant référence au schéma de relation PERSONNE.

ENSEIGNANT(id_enseignant)

Ce schéma de relation rassemble les informations concernant les enseignants.

id_enseignant est une clé étrangère faisant référence au schéma de relation PERSONNE.

MODULE(id_module, code, ue, responsable)

Ce schéma de relation rassemble les informations concernant les modules.

responsable est une clé étrangère qui fait référence au schéma de relation ENSEIGNANT

INTITULE (id_intitule, intitule_module)

Ce schéma de relation rassemble les informations concernant les intitulés des différents modules.

id_intitule est une clé étrangère qui fait référence au schéma de relation MODULE

EVALUATION (id_evaluation,id_module,nom_evaluation,date_evaluation)

Ce schéma de relation rassemble les informations concernant les évaluations.

id_module est une clé étrangère qui fait référence au schéma de relation MODULE

NOTE_EVALUATION (id_evaluation, id_etudiant, note)

Ce schéma de relation rassemble les informations concernant les notes des élèves pour chaque évaluation.

id_etudiant est une clé étrangère qui fait référence au schéma de relation ETUDIANT

id_evaluation est une clé étrangère qui fait référence au schéma de relation EVALUATION

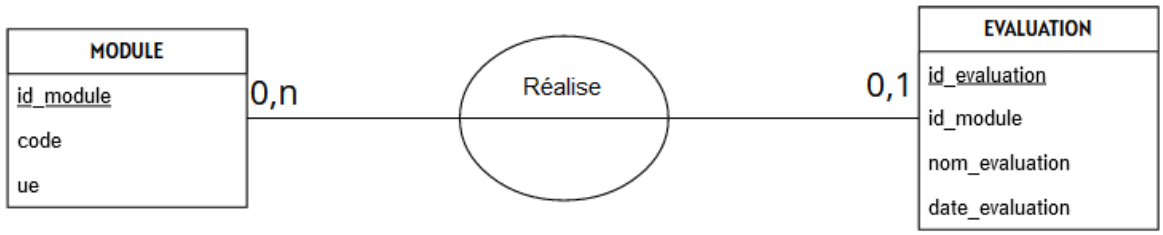
3. Script Création des Tables

```
CREATE TABLE PERSONNE (
     id_personne INTEGER PRIMARY KEY,
     prenom_personne VARCHAR,
     nom_personne VARCHAR,
);

CREATE TABLE ETUDIANT (
     id_etudiant INTEGER PRIMARY KEY REFERENCES PERSONNE(id_personne)
ON DELETE CASCADE,
);
```

```
CREATE TABLE ENSEIGNANT (
     id_enseignant   INTEGER   PRIMARY   KEY   REFERENCES   PERSONNE
(id_personne)ON DELETE CASCADE,
);

CREATE TABLE MODULE (
     id_module INTEGER PRIMARY KEY,
     code VARCHAR,
     ue VARCHAR,
);

CREATE TABLE INTITULE (
     id_intitule INTEGER PRIMARY KEY REFERENCES MODULE (id_module) ON
DELETE CASCADE,
     intitule_module VARCHAR
);

CREATE TABLE EVALUATION (
     id_evaluation SERIAL PRIMARY KEY,
     id_module REFERENCES MODULE (id_ module) ON DELETE CASCADE,
     nom_evaluation VARCHAR,
     date_evaluation DATE,
);

CREATE TABLE NOTE_EVALUATION (
      id_evaluation REFERENCES EVALUATION (id_evaluation) ,
      id_etudiant REFERENCES ETUDIANT (id_etudiant),
      note FLOAT,
      CONSTRAINT PRIMARY KEY (id_evaluation, id_module));
```

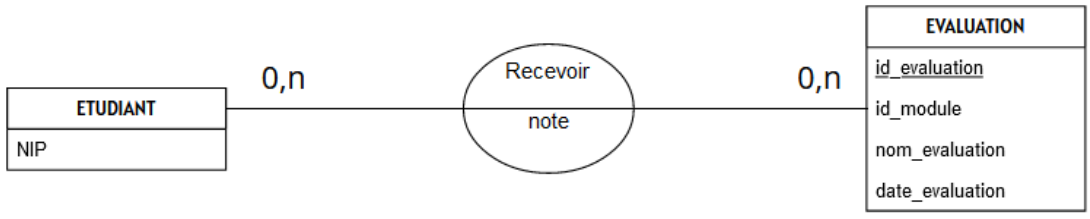
2.2 : Modélisation et script de création avec AGL

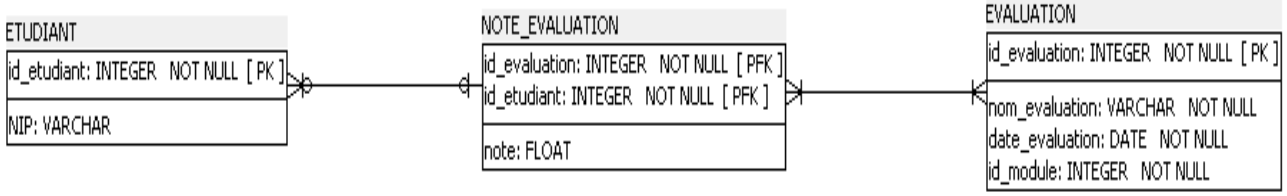
1. Illustrations comparatives cours/AGL commentée d'une association fonctionnelle



Tout d'abord, nous pouvons voir que l'Atelier de Génie Logiciel (AGL) ne représente pas les associations sous forme de bulle mais par un lien. En effet, cet AGL produit une sorte de modèle relationnel et non un modèle entité-association (MEA). Nous pouvons retrouver certaines familiarités avec notamment la clé primaire qui est à part dans l'AGL et soulignée dans le MEA. En revanche, le logiciel demande une précision, non nécessaire dans un MEA, des types (INTEGER,VARCHAR, DATE…).

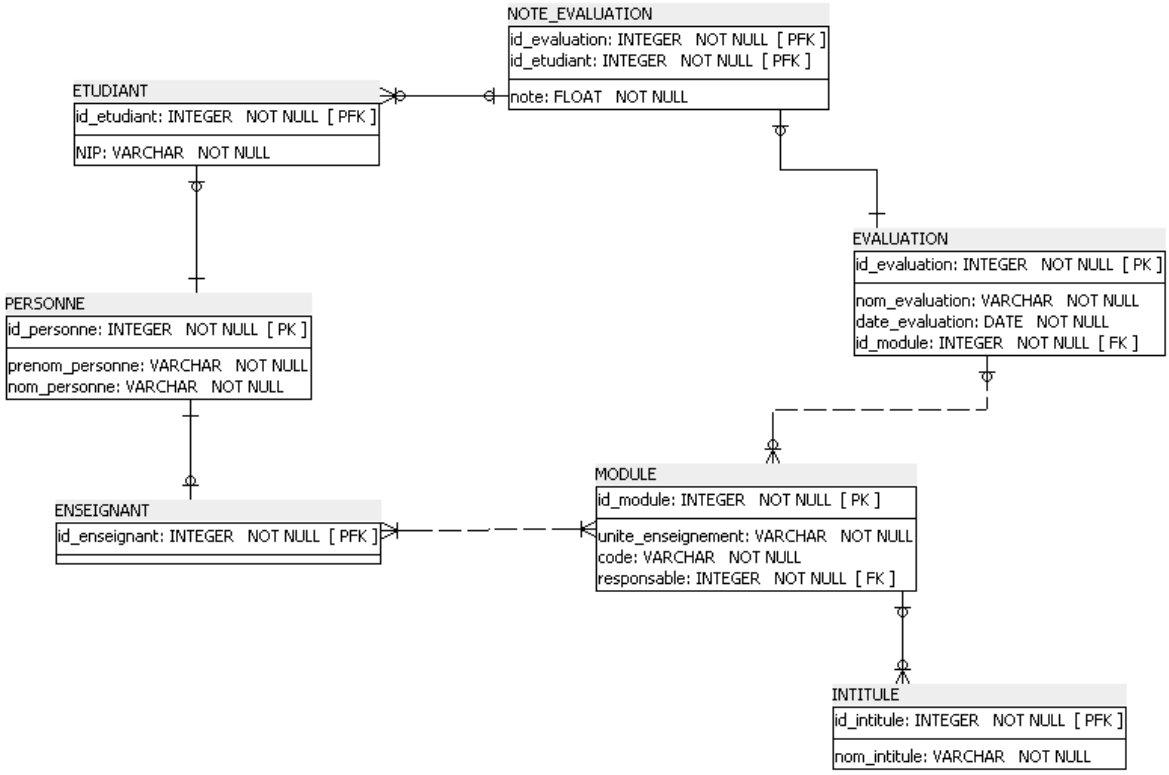
2. Illustrations comparatives cours/AGL commentée d'une association maillée





Ici, j'ai transformé l'attribut note en une table nommée NOTE_EVALUATION afin de stocker toutes les notes des étudiants. C'est pour cela que j'ai créé une clé primaire double afin qu'une évaluation corresponde à un étudiant. Cette association à la base maillée est alors devenue fonctionnelle. D'un côté nous avons 0, n et 0,1, de l'autre nous avons 1,1 et 0,1.

3. Modèle entités-associations réalisé avec l'AGL



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

```
CREATE TABLE PERSONNE (
     id_personne INTEGER NOT NULL,
     prenom_personne VARCHAR NOT NULL,
     nom_personne VARCHAR NOT NULL,
     CONSTRAINT id_personne PRIMARY KEY (id_personne)
);

CREATE TABLE ETUDIANT (
     id_etudiant INTEGER NOT NULL,
     NIP VARCHAR NOT NULL,
     CONSTRAINT id_etudiant PRIMARY KEY (id_etudiant));
```

```
CREATE TABLE ENSEIGNANT (
     id_enseignant INTEGER NOT NULL,
     CONSTRAINT id_enseignant PRIMARY KEY (id_enseignant)
);

CREATE TABLE MODULE (
     id_module INTEGER NOT NULL,
     unite_enseignement VARCHAR NOT NULL,
     code VARCHAR NOT NULL,
     responsable INTEGER NOT NULL,
     CONSTRAINT id_module PRIMARY KEY (id_module)
);

CREATE TABLE INTITULE (
     id_intitule INTEGER NOT NULL,
     nom_intitule VARCHAR NOT NULL,
     CONSTRAINT id_intitule PRIMARY KEY (id_intitule)
);

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

CREATE TABLE NOTE_EVALUATION (
     id_evaluation INTEGER NOT NULL,
     id_etudiant INTEGER NOT NULL,
     note REAL NOT NULL,
     CONSTRAINT _id_evaluation_id_module_ PRIMARY KEY (id_evaluation,
id_etudiant));
```

```
ALTER TABLE ENSEIGNANT ADD CONSTRAINT etre
FOREIGN KEY (id_enseignant)
REFERENCES PERSONNE (id_personne)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE ETUDIANT ADD CONSTRAINT etre
FOREIGN KEY (id_etudiant)
REFERENCES PERSONNE (id_personne)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE NOTE_EVALUATION ADD CONSTRAINT recevoir
FOREIGN KEY (id_etudiant)
REFERENCES ETUDIANT (id_etudiant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE MODULE ADD CONSTRAINT participe
FOREIGN KEY (responsable)
REFERENCES ENSEIGNANT (id_enseignant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

```
ALTER TABLE EVALUATION ADD CONSTRAINT realise
FOREIGN KEY (id_module)
REFERENCES MODULE (id_module)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE INTITULE ADD CONSTRAINT etreintitule
FOREIGN KEY (id_intitule)
REFERENCES MODULE (id_module)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE NOTE_EVALUATION ADD CONSTRAINT creer
FOREIGN KEY (id_evaluation)
REFERENCES EVALUATION (id_evaluation)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

5. Discussion sur les différences entre les scripts produits manuellement et automatiquement

Les scripts de création de tables sont presque les mêmes. La manière de définir une clé primaire change légèrement. Quant aux clés étrangères, manuellement nous faisons un "référencement" à la clé/attribut venant de la table parent. Or là, le script généré modifie la table afin d'ajouter la clé étrangère, après création de toutes. Je suppose que l'ordinateur crée les tables dans l'ordre dans lequel nous les avons créées. Par conséquent, les tables ne sont pas forcément créées dans un ordre précis et cela pourrait créer des erreurs lors de la création de clé étrangère qui nécessite une table parent avant d'associer avec une autre. C'est sans doute pour cela que l'AGL les ajoutent après. De plus, nous voyons que le script généré précise partout des actions en plus/moins que l'on ne précise pas forcément manuellement. Voici un exemple : Dans notre script, nous précisons les NOT NULL seulement lorsque cela est utile contrairement à l'AGL qui le précise à tous les attributs. Pour le changer, il faut l'enlever dans les paramètres de l'attribut s'il peut avoir une valeur nulle. Enfin, lors de la création des clés étrangères, le script généré rajoute des options comme ON DELETE NO ACTION,

NOT DEFERRABLE, ON UPDATE NO ACTION. Après recherche, ce sont des valeurs par défaut que l'on peut changer dans les paramètres de l'attribut encore une fois.

2.3 : Peuplement des tables et requêtes

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

Tout d'abord, j'ai créé une table data comportant tous les attributs disponibles dans le fichier data.csv donné avec la requête suivante afin de remplir notre base de données plus facilement à l'aide de requêtes SQL par la suite :

```
CREATE TABLE data (
     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
);
```

Pour remplir cette table, j'ai alors utilisé cette méta-commande qui va copier toutes les données du fichier csv et les mettre dans la table créée.

```
\copy data FROM data.csv WITH (FORMAT CSV, HEADER, DELIMITER ' ;');
```

Enfin, j'ai rempli toutes les tables avec ces requêtes :

```
INSERT INTO personne (id_personne, prenom_personne,nom_personne)
SELECT DISTINCT id_etudiant, prenom_etudiant, nom_etudiant
FROM data;


INSERT INTO personne (id_personne, prenom_personne, nom_personne)
SELECT DISTINCT id_enseignant, prenom_enseignant, nom_enseignant
FROM data;

INSERT INTO enseignant (id_enseignant)
SELECT DISTINCT id_enseignant
FROM data;

INSERT INTO etudiant (id_etudiant)
SELECT DISTINCT id_etudiant
FROM data;


INSERT INTO module(id_module, unite_enseignement, code, responsable)
SELECT DISTINCT id_module, ue, code, id_enseignant
FROM data;


INSERT INTO evaluation (nom_evaluation, date_evaluation, id_module)
SELECT nom_evaluation, date_evaluation, id_module
FROM data;


INSERT INTO intitule (id_intitule,nom_intitule)
SELECT DISTINCT id_module, nom_intitule
FROM data;
```

```
INSERT INTO note_evaluation (id_evaluation, id_etudiant, note)
SELECT DISTINCT id_evaluation, id_etudiant, note
FROM data
JOIN evaluation
ON data.nom_evaluation = evaluation.nom_evaluation
AND data.date_evaluation = evaluation.date_evaluation;
```

La base de données est maintenant complète et nous pouvons la manipuler.

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

Voici alors deux requêtes intéressantes qui nous permettent d'utiliser toutes les tables :

La 1ère requête va servir à savoir quel(s) étudiant(s) ont eu la plus mauvaise note de toute la promotion et dans quel module.

Nous allons alors afficher la plus petite note possible accompagnée du nom et prénom de la ou les personnes qui l'ont eu. Nous afficherons également l'intitulé du module. Pour cela, nous allons faire plusieurs jointures entre les tables comprenant les attributs dont nous avons besoin. Nous devrons ajouter une condition afin que la note affichée soit la plus basse de toutes les notes.

```
SELECT MIN (note) AS note_min, nom_personne, prenom_personne, nom_intitule
FROM note_evaluation
JOIN PERSONNE
ON id_etudiant = id_personne
JOIN EVALUATION
USING (id_evaluation)
JOIN MODULE
USING (id_module)
JOIN INTITULE
ON id_module = id_intitule
GROUP BY note, prenom_personne, nom_personne, nom_intitule
HAVING note = (SELECT MIN (note) FROM note_evaluation);
```

Cela nous renverra alors :

```
note_min | nom_personne | prenom_personne |            nom_intitule
----------+-------------+-----------------+---------------------------------
     0.25 | Franceschi   | Bruno           | Implémentation d'un besoin client
     0.25 | Chaix        | Edouard         | Implémentation d'un besoin client
     0.25 | Massone      | Henriette       | Initiation au développement
     0.25 | Cruz         | Herve           | Implémentation d'un besoin client
     0.25 | Marolleau    | Juste           | Initiation au développement
     0.25 | Dulac        | Kevin           | Implémentation d'un besoin client
     0.25 | Fournel      | Lucienne        | Introduction à l'architecture des ordinateurs
     0.25 | Marchand     | Michael         | Implémentation d'un besoin client
     0.25 | Bataille     | Philippe        | Implémentation d'un besoin client
     0.25 | Guelle       | Stephane        | Implémentation d'un besoin client
     0.25 | Ciolli       | Viviane         | Implémentation d'un besoin client
(11 lignes)
```

Dans cette deuxième requête, nous cherchons à savoir quel module a fait le plus d'évaluation et qui est le responsable de ce module. Ainsi, nous afficherons le nom et le prénom du responsable, l'intitule du module et le nombre d'évaluation qui a été faite. Pour cela, de la même manière qu'avant nous allons faire une jointure entre plusieurs tables qui comportent les informations dont on a besoin. Afin d'avoir les nombres d'évaluation effectuées, nous devons faire une selection dans la clause FROM. Ainsi, nous agirons sur les nombres d'évaluation. Puis, pour sélectionner le plus haut nombre, nous devrons l'indiquer à la fin.

```
SELECT prenom_personne, nom_personne, nom_intitule, nb_eval

FROM (SELECT COUNT(id_evaluation)AS nb_eval, id_module
     FROM EVALUATION
     WHERE id_module =  id_module
     GROUP BY id_module)
     AS EVAL_MODULE
JOIN intitule
ON id_module = id_intitule
JOIN MODULE

USING (id_module)
JOIN PERSONNE
ON responsable = id_personne
GROUP BY prenom_personne, nom_personne, nom_intitule, nb_eval
HAVING nb_eval = (SELECT MAX(nb_eval)
      FROM (SELECT COUNT(id_evaluation) AS nb_eval, id_module
      FROM EVALUATION
      WHERE id_module = id_module
      GROUP BY id_module)
      AS EVAL_MODULE

);
```

Et cela nous renverra :

```
prenom_personne  | nom_personne |        nom_intitule          | nb_eval
-----------------+--------------+------------------------------+---------
 Anne            | Heron        | Initiation au développement |       8
(1 ligne)
```