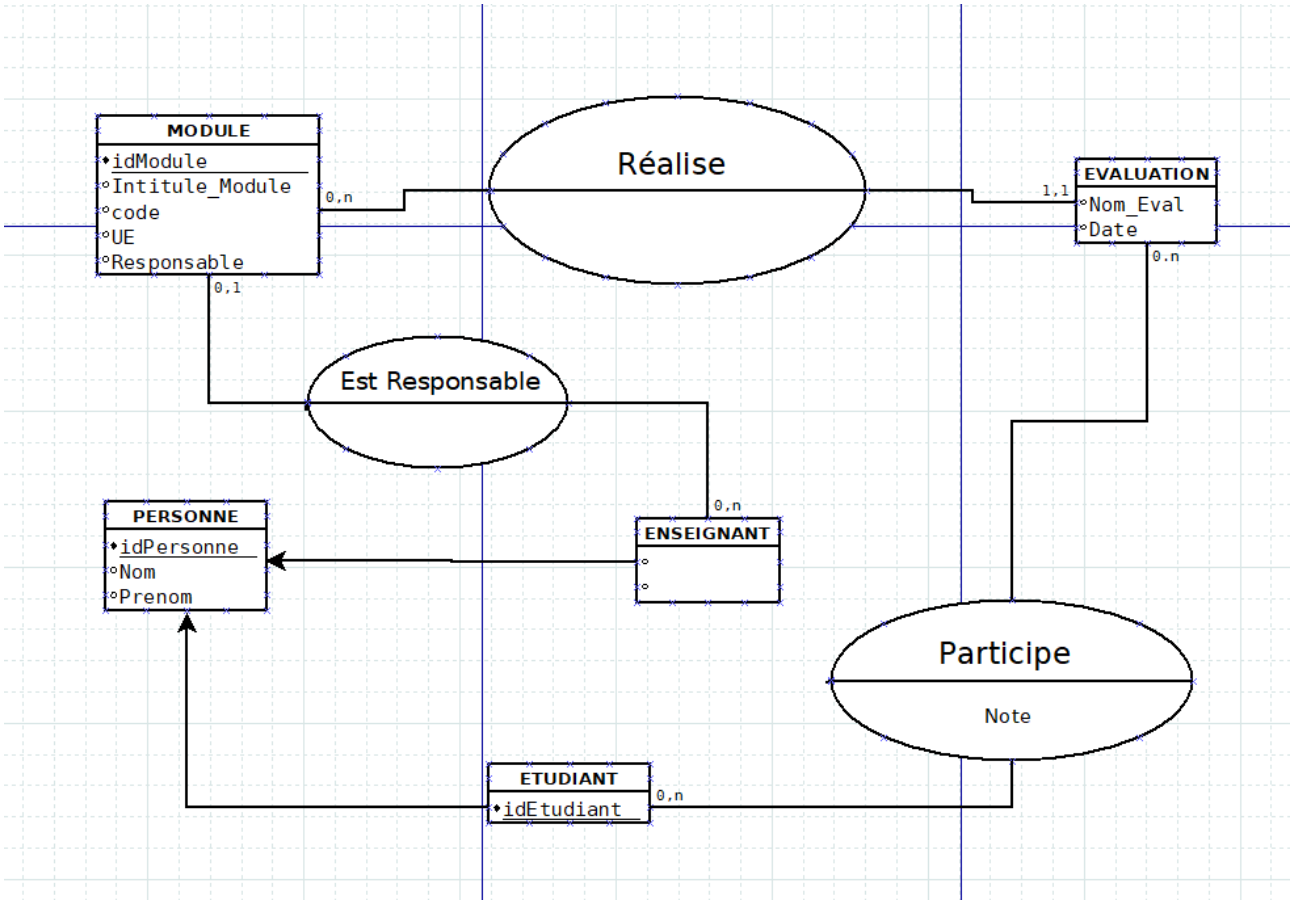
RAPPORT BDD

LI
Loïc
Tlaloc

2.1 Modélisation et script de création « sans AGL »

1/ Modèle entité-associations :



2/ Schéma relationnel :

PERSONNE(idPersonne, nom, prénom)
MODULE(idModule, intitulé_Module, Code, UE)
Enseignant(idPersonne*)
ÉTUDIANT(idEtudiant, idPersonne*, note)
ÉVALUATION(Nom_Évaluation, Date)

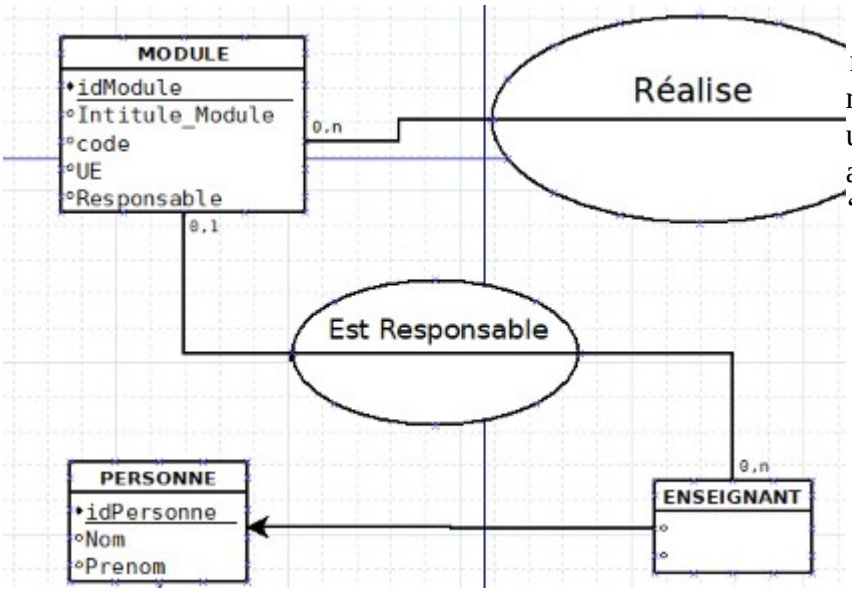
3/ Scripts SQL de création de table :

```
CREATE TABLE personne (
        idPersonne INTEGER PRIMARY KEY,
        nom VARCHAR NOT NULL,
        prenom VARCHAR NOT NULL
);

CREATE TABLE enseignant (
        id_enseignant REFERENCES personne (id_personne),
);

CREATE TABLE module (
        id_module INTEGER PRIMARY KEY,
        intitule_module VARCHAR NOT NULL,
        code VARCHAR NOT NULL,
        ue VARCHAR NOT NULL,
        responsable REFERENCES (personne id_enseignant)
);

CREATE TABLE etudiant (
        id_etudiant INTEGER PRIMARY KEY,
        id_personne REFERENCES personne (id_personne)
        note FLOAT
);

CREATE TABLE evaluation (
        nom_evaluation VARCHAR NOT NULL,
        DATE
);
```

2.2 Modélisation et script de création « avec AGL »

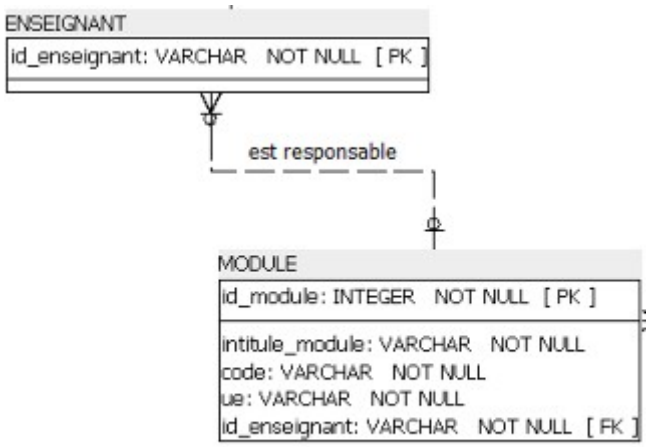
1/ Une association fonctionnel du cours est différente de celui avec l'AGL, car les liaisons ce font par des symboles au bout de chaque côté qui represente la cardinalité donc on ne retrouve plus les cardinalités chiffrés.
Sur l'AGL on retrouve pas les type association et les clés sont automatiquement mise dans la table relier par un lien.

Exemple de type association fonctionnel avec AGL et sans AGL :



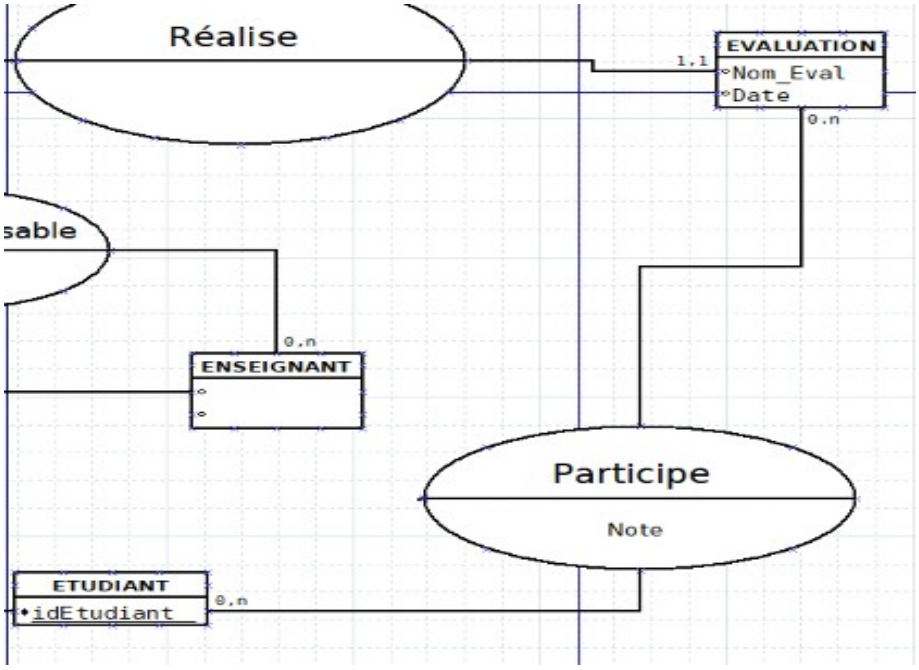
relation « est responsable » est fonctionnel car une relation fonctionnel est une association de cardinalité de max '1' vers max 'n'.

Relation entité-associations fonctionnel sur AGL :



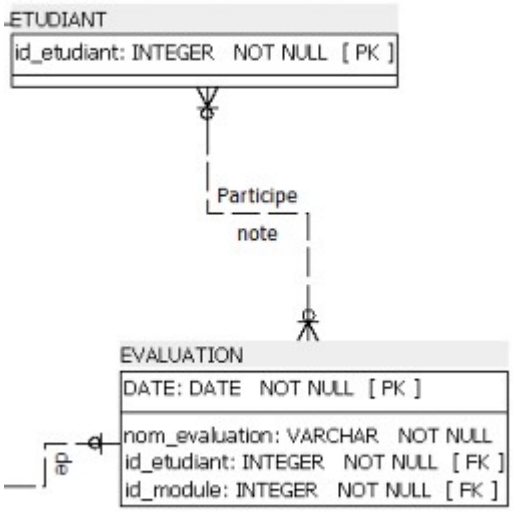
2/ Pour l'association maillé, on se retrouve un peu près dans le même cas qu'avec une association fonctionnel, les cardinalités sont remplacer par des symboles et on ne retrouve pas de type association.

Exemple de type association maillé avec AGL et sans AGL :

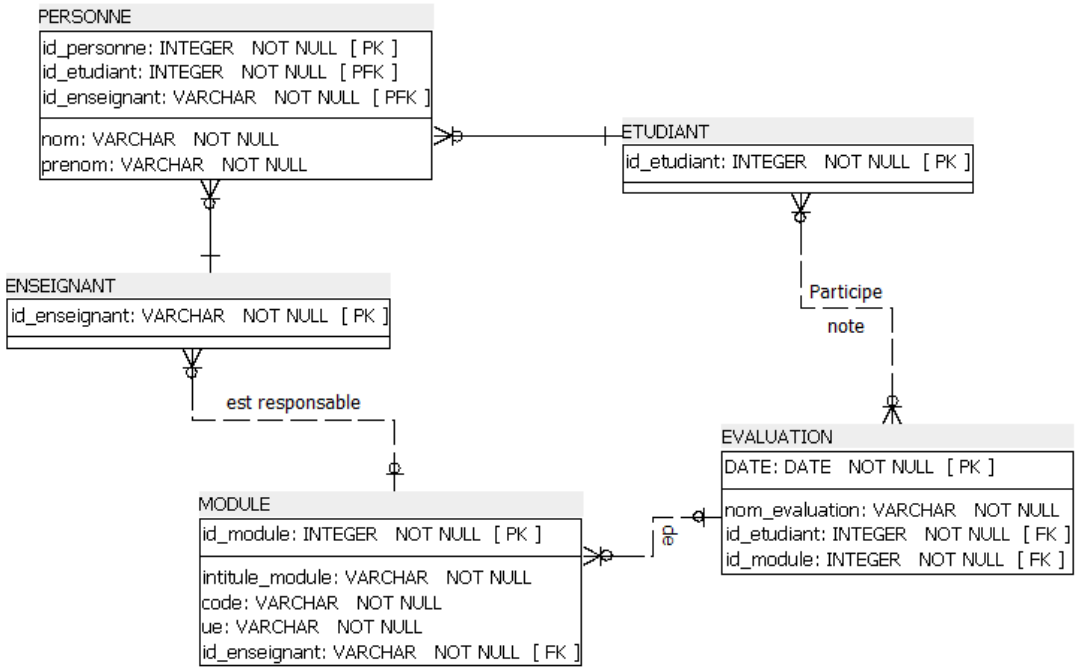


La relation association 'Participe' est maillé car on a une cardinalité de max 'n' vers max 'n'.

Le type association maillé créer sur l'AGL.



3/ Modélisation entité-associations AGL :



4/ Le script générer avec ce modèle entité-associations :

```
CREATE TABLE public.PERSONNE (
	id_personne INTEGER NOT NULL,
	nom VARCHAR NOT NULL,
	prenom VARCHAR NOT NULL,
	CONSTRAINT id_personne PRIMARY KEY (id_personne)
);

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

CREATE TABLE public.ENSEIGNANT (
```

```
	id_personne INTEGER NOT NULL,
	id_enseignant VARCHAR NOT NULL,
	CONSTRAINT id_personne PRIMARY KEY (id_personne, id_enseignant)
);


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


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




ALTER TABLE public.ENSEIGNANT ADD CONSTRAINT personne_enseignant_fk
FOREIGN KEY (id_personne)
REFERENCES public.PERSONNE (id_personne)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

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

ALTER TABLE public.EVALUATION ADD CONSTRAINT etudiant_evaluation_fk
FOREIGN KEY (id_etudiant)
REFERENCES public.ETUDIANT (id_etudiant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE public.MODULE ADD CONSTRAINT enseignant_module_fk
FOREIGN KEY (id_enseignant)
```

```
REFERENCES public.ENSEIGNANT (id_enseignant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

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

5/ Différence entre les tables créer par l'AGL et celui réaliser manuellement :

Dans le script réaliser par l'AGL, on a d'abord la creation de toute les table puis des commandes avec « ALTER TABLE » alors que manuellement on a seulement « CREATE TABLE », y a aussi la partie du nom de table qui commence par un « public. »nom_table alors que manuellement on a que le nom de la table.

Puis le script générer par l'AGL c'est plus complet avec les restrictions et les clés étrangers.
Et le script est plus long et moins compréhensible que celui fait manuellement. Sinon celui générer par l'AGL est plus simple à réaliser puisque, nous on a juste a faire le schéma relationnel donc pas besoin de schéma relationnel et de faire le script sois-même.

En conséquence si, notre entité-associations est faux alors le script aurai aussi des problèmes.

2.3 Peuplement des tables et requête :

1/ Le script créer pour peuplé la base de donnée et les tables :

On commence par supprimer les tables qu'on va utiliser pour créer et remplir les tables

```
DROP TABLE imp ;
```

```
DROP TABLE module ;
DROP TABLE evaluation ;
DROP TABLE personne ;
DROP TABLE etudiant ;
DROP TABLE enseignant ;
```

Puis, commence par créer une table temporaire qui va permettre de stocker toutes les colonnes/donnée du fichier csv. Dans cette table, on doit retrouver les mêmes noms de colonne.

```
CREATE TABLE imp
(
	id_enseignant integer,
	nom_enseignant character varying,
	prenom_enseignant character varying,
	id_module integer,
	code character varying,
	ue character varying,
	intitule_module character varying,
	nom_evaluation character varying,
	date_evaluation date, note numeric,
	id_etudiant integer,
	nom_etudiant character varying,
	prenom_etudiant character varying
)
```

Cette ligne permet de copier dans la table créer au-dessus les différentes colonnes dans le fichier .csv.

```
COPY imp
FROM 'C:\Program Files\PostgreSQL\15\scripts\data.csv'
DELIMITER ';' CSV Header;
```

On commence par créer la table enseignant, puis utilise la fonction INSERT INTO pour pouvoir ajouter a la table enseignant tout les id_enseignant qui se retrouve dans la table ‘imp’.

```
CREATE TABLE enseignant
(
	id_enseignant integer NOT NULL
)

INSERT INTO enseignant (id_enseignant) SELECT id_enseignant FROM imp;
```

Création de la table etudiant avec l’ajout de l’id_etudiant ainsi que la note obtenue par l’etudiant qui passe une evaluation d’un module.

```
CREATE TABLE etudiant
(
    id_etudiant integer NOT NULL,
    note numeric NOT NULL
)

INSERT INTO etudiant (id_etudiant, note) SELECT id_etudiant, note FROM imp;
```

Création de la table personne, on ajoute d'abords tout les id_etudiant avec les nom et prenom etudiant puis on rajoute a la table personne les id_enseignant, nom et prenom des enseignant. Dans la table personne, on retrouve seulement id_personne, nom et prenom, donc les id_personne font références aux id_etudiant et id_enseignant et pareil pour les prénoms et les noms.

```
CREATE TABLE personne
(
    id_personne integer,
    nom VARCHAR,
    prenom VARCHAR
)

INSERT INTO personne (id_personne, nom, prenom) SELECT id_etudiant, nom_etudiant,
prenom_etudiant FROM imp;
INSERT INTO personne(id_personne, nom, prenom) SELECT id_enseignant,
nom_enseignant, prenom_enseignant FROM imp;
```

Création de la table evaluation avec les attributs nom_evaluation et date_evaluation.

```
CREATE TABLE evaluation
(
    nom_evaluation VARCHAR,
    date_evaluation DATE
)

INSERT INTO evaluation (nom_evaluation, date_evaluation) SELECT nom_evaluation,
date_evaluation FROM imp;
```

Et enfin la création de la table module qui aura plusieur attribut:id_module, intitule_module, code, ue et le responsable du module.
Tout les attribut sont prise a partir de la table temporaire 'imp' sauf responsable qui fait références aux id_enseignant qui va justement dire le nom et prenom de l'enseignant qui est le responsable du module.

```
CREATE TABLE module
(
    id_module integer NOT NULL,
    intitule_module character varying,
    code character varying,
    ue character varying,
    responsable character varying
)
```

INSERT INTO module (id_module, intitule_module, code, ue, responsable) SELECT id_module, intitule_module, code, ue, id_enseignant FROM imp;

Après que tout est finit on peut supprimer la table temporaire

DROP TABLE imp

2/

« SELECT DISTINCT id_personne,nom, prenom FROM personne JOIN enseignant ON personne.id_personne = enseignant.id_enseignant ORDER BY id_personne; »
Cette commande va permet d'afficher tout les enseignant avec leur id, nom et prenom sans répétition.
Car avec un « SELECT id_personne, nom, prenom FROM personne JOIN enseignant ON personne.id_personne = enseignant.id_enseignant ; », on se retrouve avec un erreur et affiche la même personne des milliers de fois. Donc avec un DISTINCT après SELECT on aura les différentes personnes.

« SELECT DISTINCT id_personne, nom, prenom FROM personne JOIN etudiant ON personne.id_personne = etudiant.id_etudiant; »
Cette fois, la commande affiche les etudiants, leur id, nom et prénom.