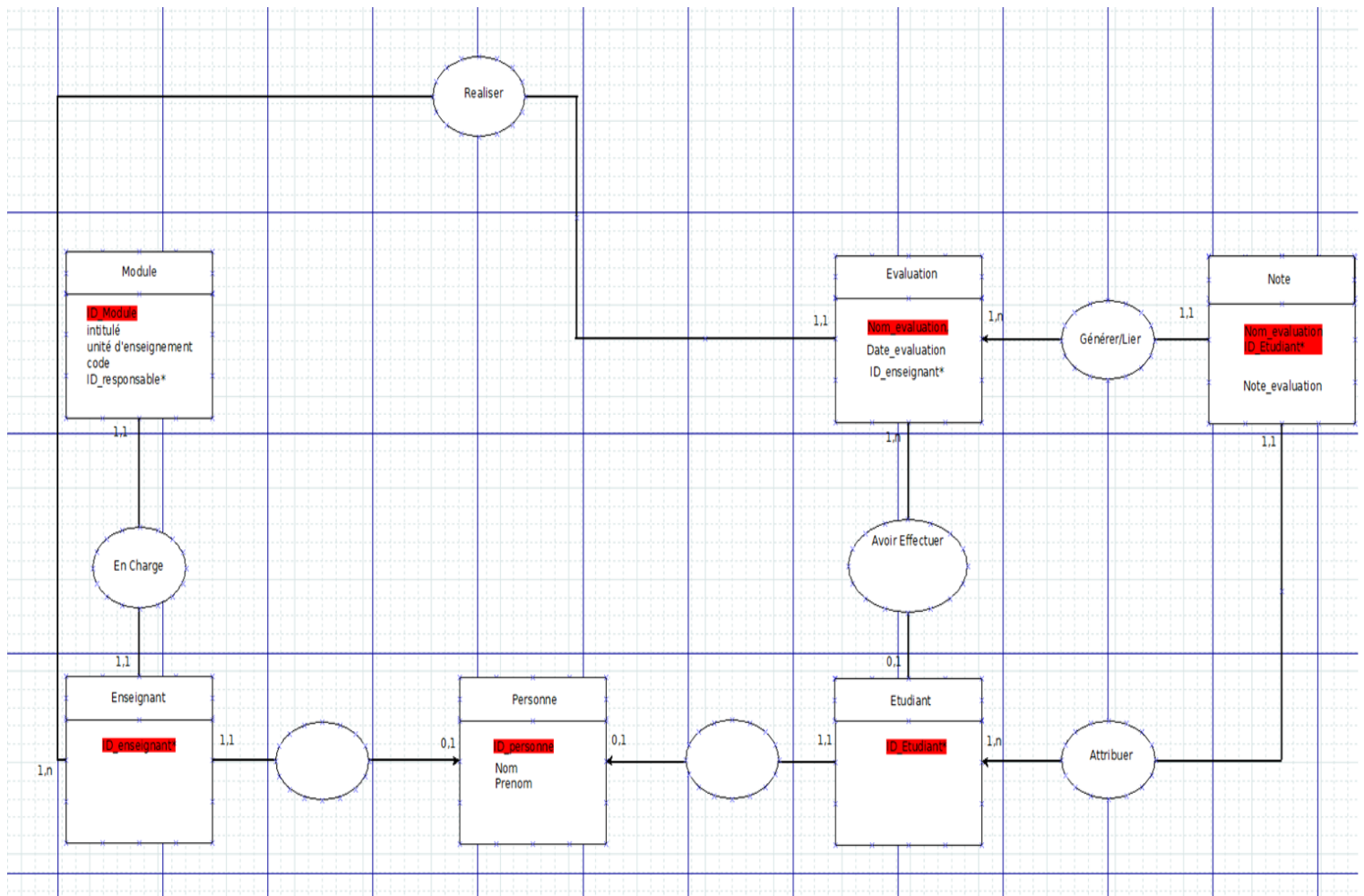


SAE104-BDD

2.1 Modélisation et script de création «sans AGL»

1. Modèle entités-association



2. Schéma relationnel

Schéma relationnel :

-personne (id_personne, nom, prenom)

-enseignant(id_enseignant) où id_enseignant est à la fois clé primaire du schéma de relation enseignant, et une clé étrangère qui fait référence au schéma de relation de personne.

-etudiant(id_etudiant) où id_etudiant est à la fois clé primaire du schéma de relation étudiant ,et une clé étrangère qui fait référence au schéma de relation de personne.

-module(id_module, intitulé, unité d'enseignement, code, id_responsable) où id_responsable est une clé étrangère qui fait référence au schéma de relation de enseignant.

-evaluation(nom_evaluation, date_evaluation, id_enseignant) où id_enseignant est une clé étrangère qui fait référence au schéma de relation de enseignant.

-note (nom_evaluation, id_etudiant, note_evaluation) où nom_evaluation et id_etudiant sont à la fois clé primaire de la relation note, et clé étrangère qui font respectivement référence aux schéma de relation etudiant et evaluation.

3. Script SQL de création des tables

Table personne:

```
CREATE TABLE personne
(
    id_personne INTEGER PRIMARY KEY,
    nom VARCHAR,
    prenom VARCHAR
);
```

Table enseignant :

```
CREATE TABLE enseignant
(
    id_enseignant INTEGER REFERENCES
personne(id_personne),
    PRIMARY KEY(id_enseignant)
);
```

Table etudiant :

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

```
CREATE TABLE module
(
    id_responsable INTEGER REFERENCES
enseignant(id_enseignant),
    id_module INTEGER PRIMARY KEY,
    ue VARCHAR,
    code VARCHAR,
    intitule VARCHAR
);
```

Table evaluation :

```
CREATE TABLE evaluation
(
    id_enseignant INTEGER REFERENCES
enseignant(id_enseignant),
    nom_evaluation VARCHAR PRIMARY KEY,
    date_evaluation DATE
);
```

Table note :

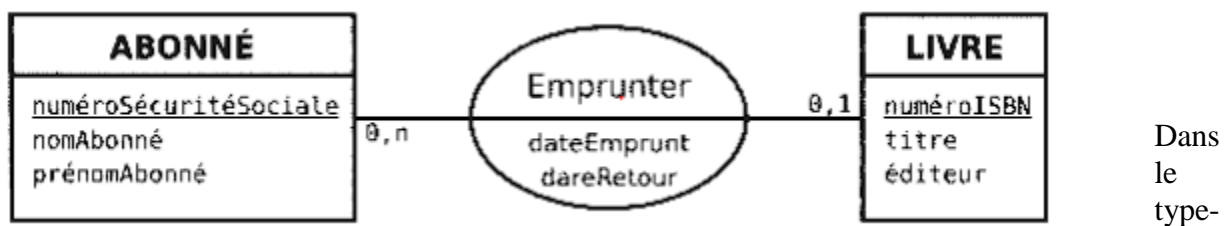
```
CREATE TABLE note
(
    id_etudiant INTEGER REFERENCES etudiant(id_etudiant),
    nom_evaluation VARCHAR REFERENCES
evaluation(nom_evaluation),
    note_evaluation FLOAT,
    PRIMARY KEY(nom_evaluation, id_etudiant)
);
```

2.2 Modélisation et script de création « Avec AGL »

1. type-association fonctionnel AGL



type-association fonctionnel Cours

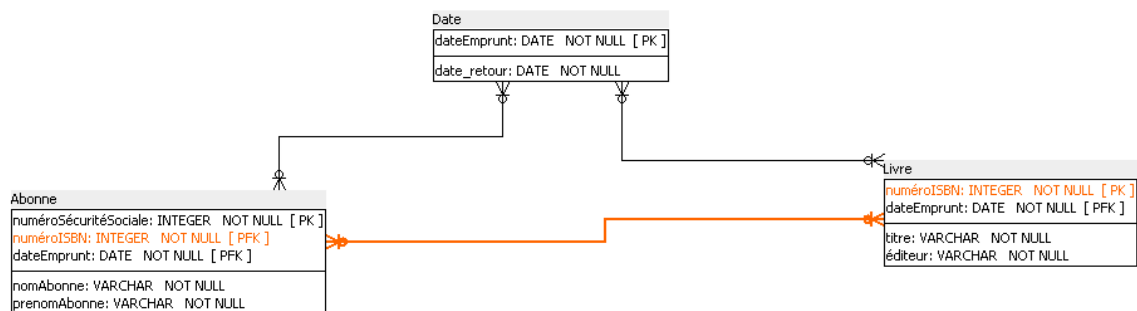


association AGL les type-association ne sont pas visible contrairement à la version du cours, les cardinalités ne sont visibles que par des schémas de liens (flèches qui lient les types entités) différents, alors que dans le cours les cardinalités sont représentées par 0,1 et n, de plus les clé primaire et étrangère sont spécifier et voyante dans les autres tables associé.

Dans la version AGL le type des attributs est spécifié (Varchar, Integer), la syntaxe est différente de celle vue en cours, de plus vue qu'il n'y a pas de type-association dans la version AGL, les 2 types-entité ne sont pas mis en relation car ils n'ont pas de clé étrangère.

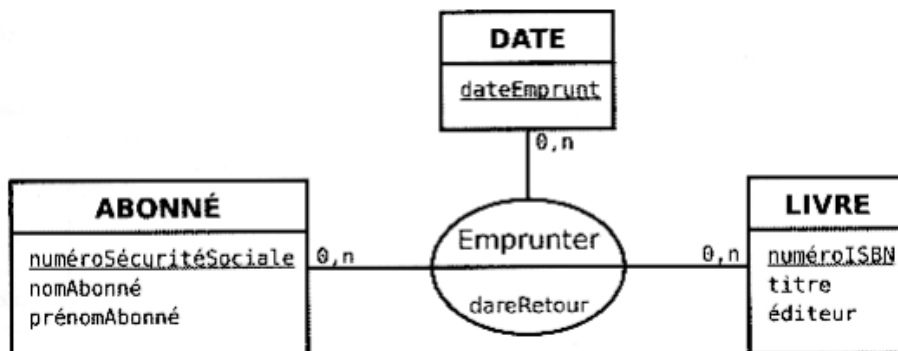
2.2

Types-association maillé AGL



Types-association maillé Cours

nous obtenons le modèle entité-association suivant :



C'est globalement comme la 2.1 les différences ne se trouvent que dans la syntaxe, types-association pas représenté ...

4. Modèle entités-associations avec AGL


```
code VARCHAR NOT NULL,  
id_responsable INTEGER,  
id_enseignant INTEGER NOT NULL,  
CONSTRAINT module_pkey PRIMARY KEY (id_module)  
);
```

```
CREATE TABLE public.etudiant (  
    id_etudiant INTEGER NOT NULL,  
    CONSTRAINT etudiant_pkey PRIMARY KEY (id_etudiant)  
);
```

```
CREATE TABLE public.note (  
    nom_evaluation VARCHAR NOT NULL,  
    id_etudiant INTEGER NOT NULL,  
    note_evaluation REAL NOT NULL,  
    CONSTRAINT note_pkey PRIMARY KEY (nom_evaluation, id_etudiant)  
);
```

```
ALTER TABLE public.etudiant ADD CONSTRAINT etudiant_id_etudiant_fkey  
FOREIGN KEY (id_etudiant)  
REFERENCES public.personne (id_personne)  
ON DELETE NO ACTION  
ON UPDATE NO ACTION  
NOT DEFERRABLE;
```

```
ALTER TABLE public.enseignant ADD CONSTRAINT  
enseignant_id_enseignant_fkey  
FOREIGN KEY (id_enseignant)  
REFERENCES public.personne (id_personne)  
ON DELETE NO ACTION  
ON UPDATE NO ACTION  
NOT DEFERRABLE;
```

```
ALTER TABLE public.module ADD CONSTRAINT module_id_responsable_fkey  
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  
evaluation_id_enseignant_fkey
```

```
FOREIGN KEY (id_enseignant)
REFERENCES public.enseignant (id_enseignant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

```
ALTER TABLE public.note ADD CONSTRAINT note_nom_evaluation_fkey
FOREIGN KEY (nom_evaluation)
REFERENCES public.evaluation (nom_evaluation)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

```
ALTER TABLE public.note ADD CONSTRAINT note_id_etudiant_fkey
FOREIGN KEY (id_etudiant)
REFERENCES public.etudiant (id_etudiant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

5.

Le script généré par l'AGL est beaucoup plus complet que le mien, que ce soit pour les clé primaires/secondaires au niveau des contraintes , au niveau du type des clé, la suppression des lignes nuls , etc.

2.3 Peuplement des tables et requêtes

1. J'ai fais par morceau le script de peuplement en m'inspirant du fichier csv et je l'ai adapter a ma bdd pour vérifier son bon fonctionnement, pour cela j'ai réalisé des fichier .txt pour chacune des tables, que j'ai projeter à l'aide de la commande \copy <table> FROM <fichier>.

Exemple de mes fichier.txt :

evaluation - Bloc-notes

Fichier	Edition	Format	Affichage	Aide
145	Controle moyen 2- Bonus mini controle 4	27/10/21		
161	Contrôle 1	24/11/21		
157	Mini-test 3 Logique	08/11/21		
150	Oral presentation	20/01/22		
157	Mini-test 2 Relations	04/10/21		
145	Controle moyen	29/09/21		
144	Evaluation phase 1	06/09/21		
146	Contrôle court	15/12/21		
157	Mini-test 21 Relations	04/10/21		

*note - Bloc-notes

Fichier	Edition	Format	Affichage	Aide
124	Controle moyen 2- Bonus mini controle 4	6		
37	Contrôle 1	7.25		
7	Mini-test 3 Logique	6		
51	Contrôle Ensembles et relations	12.16		
126	Oral presentation	15.5		
43	Mini-test 2 Relations	10		
131	Evaluation phase 1	2.3		
123	Contrôle court	3		

(il est possible que certaines infos ne soit pas visible dans mes fichiers de remplissage, car j'en ai ajouter certaines au fur et à mesure)

```

pgsql -U postgres

postgres=# SELECT nom, prenom, note_evaluation FROM etudiant JOIN personne ON id_personne=id_etudiant JOIN note ON etudiant.id_etudiant=note_id_etudiant WHERE note_evaluation >3 ;
 nom | prenom | note_evaluation
-----+-----+-----
Calamel | Claudius | 6
Havez | Catherine | 7.25
Schilling | Marius | 6
Canet | Georgette | 12.16
Chaumaz | Herve | 15.5
Bagur | Gerard | 10
Carron | Oceane | 17.75
Dangreaux | Ngoc | 9.5
8 lignes)

postgres=# SELECT* FROM etudiant JOIN personne ON id_personne=id_etudiant ;
 id_etudiant | id_personne | nom | prenom
-----+-----+-----+-----
124 | 124 | Calamel | Claudius
37 | 37 | Havez | Catherine
7 | 7 | Schilling | Marius
51 | 51 | Canet | Georgette
126 | 126 | Chaumaz | Herve
43 | 43 | Bagur | Gerard
63 | 63 | Carron | Oceane
131 | 131 | Delayen | Cindy
123 | 123 | Badji | Elodie
115 | 115 | Dangreaux | Ngoc
10 lignes)

```

J'ai effectuer une requête dans laquelle je demande à ma bdd les étudiant qui ont eu + de 3 à une évaluation, cela me permet de voir que ma bdd fais bien la différence entre les étudiants et les enseignants, et qu'elle sait aussi faire la différence entre les différents notes/évaluations.

```

postgres=# SELECT * FROM enseignant JOIN personne ON id_personne=id_enseignant ;
 id_enseignant | id_personne | nom | prenom
-----+-----+-----+-----
145 | 145 | Heron | Anne
161 | 161 | Coignard | Charles
157 | 157 | Donizau | Leon
150 | 150 | Gervais | Vincent
144 | 144 | Helin | Mohamed
146 | 146 | Denis | Olivier
154 | 154 | Rotsztein | Nicolas
(7 lignes)

postgres=# SELECT * FROM personne JOIN enseignant ON id_personne=id_enseignant JOIN module ON id_enseignant=id_responsable ;
 id_personne | nom | prenom | id_enseignant | id_responsable | id_module | intitule | ue | code
-----+-----+-----+-----+-----+-----+-----+-----+-----
145 | Heron | Anne | 145 | 145 | 2 | R101 | UE12 | Initiation au developpement
161 | Coignard | Charles | 161 | 161 | 18 | R107 | UE12 | Outils mathematiques fondamentaux
157 | Donizau | Leon | 157 | 157 | 14 | R106 | UE12 | Mathematiques discrètes
150 | Gervais | Vincent | 150 | 150 | 7 | R110 | UE12 | Anglais technique
144 | Helin | Mohamed | 144 | 144 | 1 | R104 | UE13 | Creation d'une base de données
146 | Denis | Olivier | 146 | 146 | 3 | R102 | UE12 | Developpement d'interfaces web
(6 lignes)

postgres=# SELECT * FROM personne JOIN enseignant ON id_personne=id_enseignant JOIN module ON id_enseignant=id_responsable WHERE ue='UE13' ;
 id_personne | nom | prenom | id_enseignant | id_responsable | id_module | intitule | ue | code
-----+-----+-----+-----+-----+-----+-----+-----+-----
144 | Helin | Mohamed | 144 | 144 | 1 | R104 | UE13 | Creation d'une base de données
(1 ligne)

postgres=#

```

Cette seconde requête me permet de voir que les enseignant son bien assigné à leur module, et que ma bdd sais faire la différence entre les différents responsable de module.

(ps, j'ai interverti certaine données lors du remplissage)

