SAE 1.04

Création d'une base de donnée : Notation

Latif Marya : Shango

Sommaire

1.Modélisation et script de création “sans AGL“

Modèle entités-associations

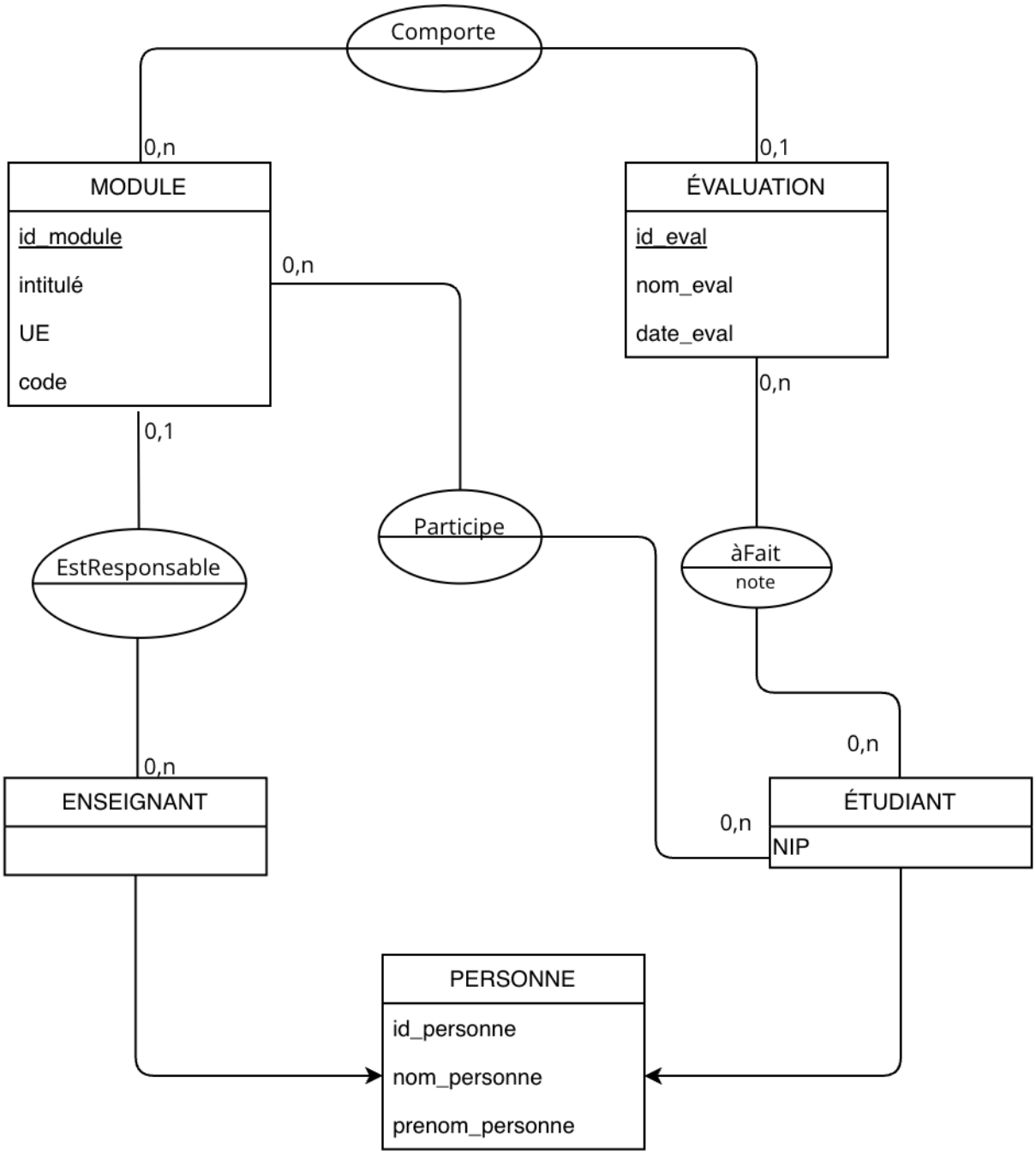


Schéma relationnel

- PERSONNE (id_personne, nom_personne, prenom_personne)
- ETUDIANT (NIP)
  - NIP fait référence à PERSONNE
- ENSEIGNANT (idEnseignant)
  - idEnseignant fait référence à PERSONNE
- MODULE (id_module, intitulé, UE, code, idEnseignant)
  - idEnseignant fait référence à ENSEIGNANT
- EVALUATION (id_eval, nom_eval, date_eval, idCours)
  - idCours fait référence à MODULE
- Participe (idModule, idEtudiant)
  - idModule et idEtudiant font référence à MODULE et ÉTUDIANT
- aFait (idEval, idEtud, note)
  - idEval et idEtud font référence à ÉVALUATION et ÉTUDIANT

Script SQL de création des tables :

```sql
CREATE TABLE Personne(

   id_personne INTEGER PRIMARY KEY,

   nom_personne VARCHAR,

   prenom_personne VARCHAR

);



CREATE TABLE Enseignant(

   id_enseignant INTEGER PRIMARY KEY REFERENCES Personne(id_personne)

);



CREATE TABLE Etudiant(

   NIP INTEGER PRIMARY KEY REFERENCES Personne(id_personne)
```

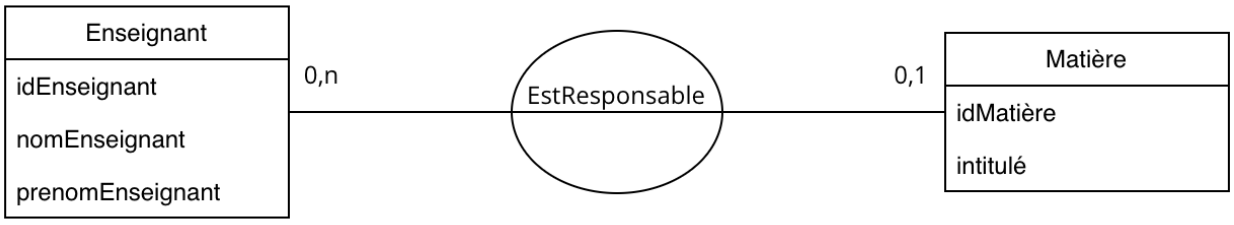
```
);


CREATE TABLE Module(
   id_module INTEGER PRIMARY KEY,
   intitule VARCHAR,
   code VARCHAR,
   UE VARCHAR,
   id_enseignant INTEGER REFERENCES Enseignant(id_enseignant)
);


CREATE TABLE Participe(
   id_module INTEGER REFERENCES Module(id_module),
   id_etudiant INTEGER REFERENCES Etudiant(NIP),
   PRIMARY KEY(id_module, id_etudiant)
);


CREATE TABLE Evaluation(
   id_eval SERIAL PRIMARY KEY,
   nom_eval VARCHAR,
   date_eval VARCHAR,
   id_module INTEGER REFERENCES MODULE(id_module)
);


CREATE TABLE aFait(
```

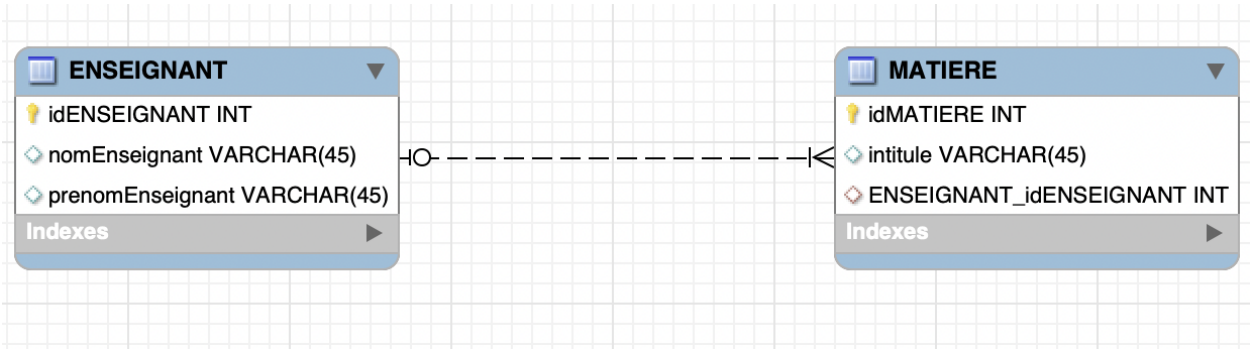
```
   id_eval INTEGER REFERENCES Evaluation(id_eval),

   id_etudiant INTEGER REFERENCES Etudiant(NIP),

   PRIMARY KEY(id_eval, id_etudiant),

   note VARCHAR

);
```

2. Modélisation et script de création “avec AGL“

Comparaison modélisation Cours/AGL

Association fonctionnelle :





La première différence que l’on peut constater est le design des tables. Dans la modélisation AGL les couleurs utilisées permettent d’avoir une meilleure visibilité et ainsi de faciliter la compréhension de celles-ci. Tandis que la version monotone du cours est moins agréable.

De plus, les symboles utilisés dans la version AGL permettent de connaître les contraintes des attributs. En effet, les clés primaires sont représentées par des clés jaunes tandis que dans la version du cours elles sont juste soulignées.

Les attributs de table sont illustrés par des losanges bleus et les références par des losanges violets ce qui permet de faire une distinction directe entre les deux, or cette distinction n'existe pas dans la version du cours.

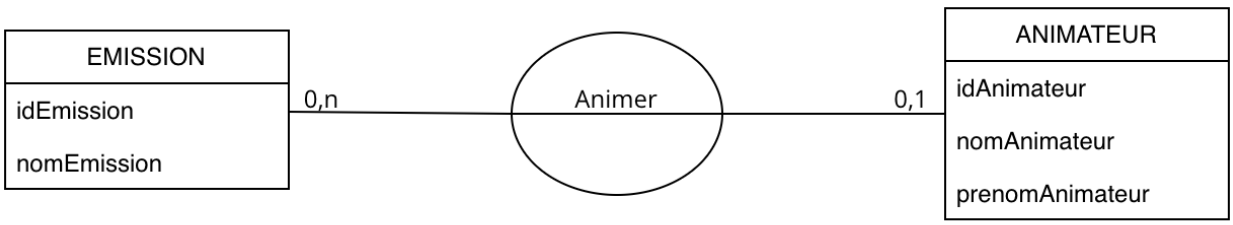
Concernant les clés étrangères, dans la version AGL, elles sont visibles dans la table mais aussi dans l'index (avec le détails des contraintes de colonnes) tandis que dans la version du cours elles ne sont pas du tout représentées.

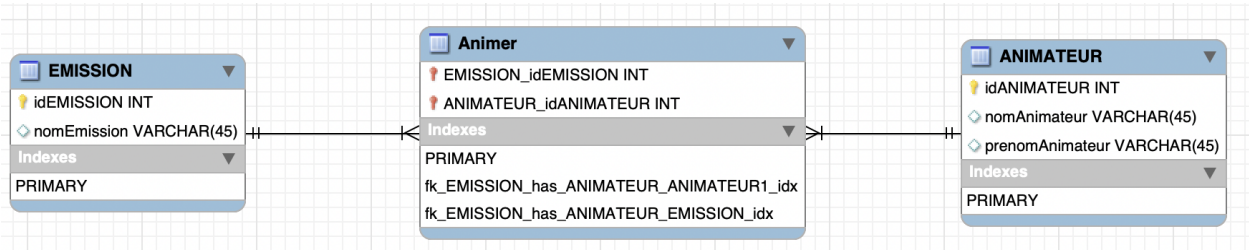
Un des points importants dans une base de données sont les types des attributs, or ces derniers figurent dans la version AGL et non dans celle du cours.

De plus, la cardinalité est illustrée différemment, dans celle du cours on indique la cardinalité minimale et maximale tandis que dans l'AGL seulement la cardinalité maximale est représentée, ce qui à mon sens est plus fidèle à ce qui est techniquement implémentable en base de données.

La représentation de l'entité d'association est également différente entre les deux modélisations. En effet, dans celle du cours elle est modélisée par une bulle nommant la relation et située entre les deux tables tandis que dans la version AGL la relation est représentée par une ligne en pointillé ce qui permet d'aérer le schéma.

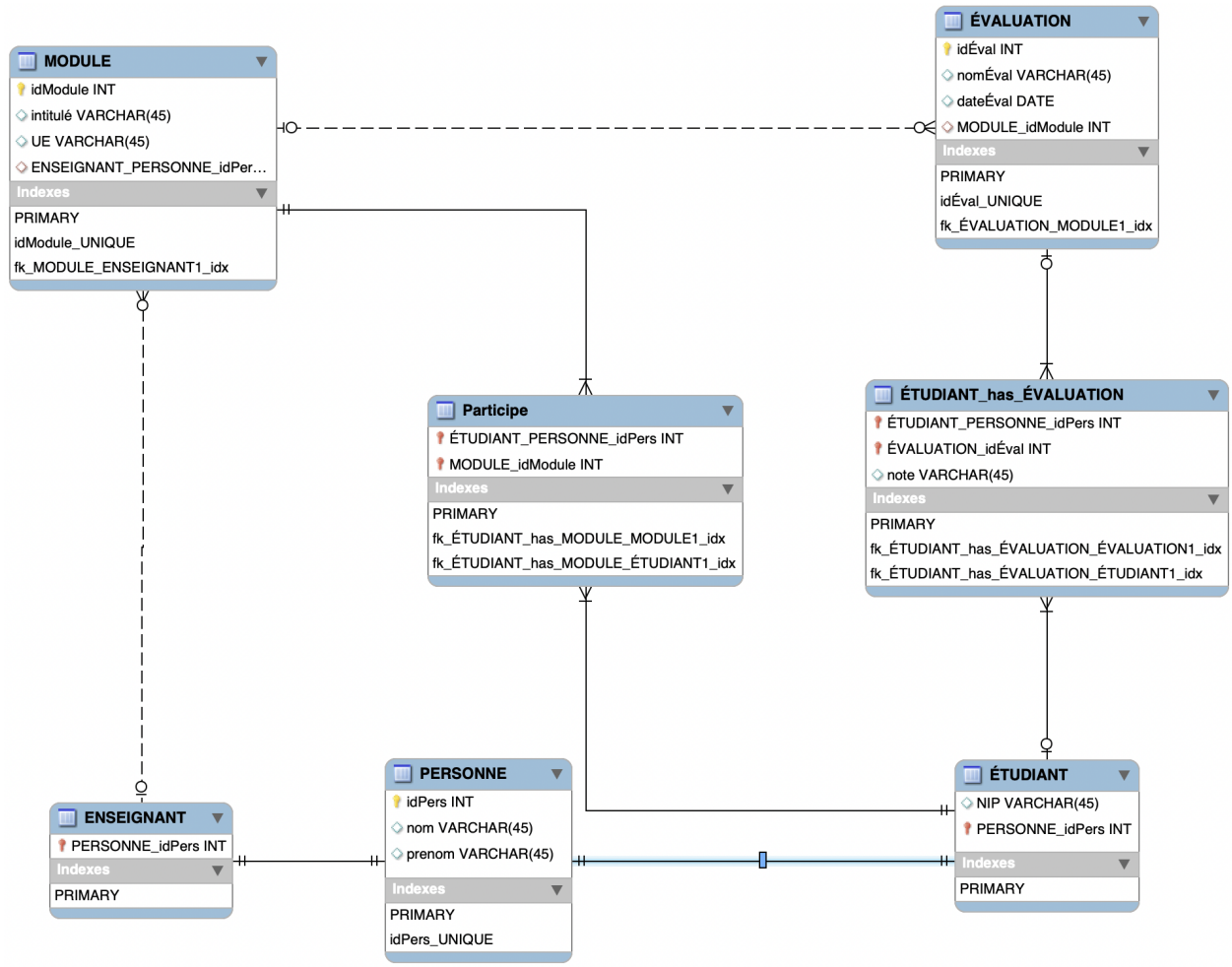
Association maillée :





En plus des comparaisons vues précédemment, dans la version AGL lorsque la relation est maillée, la table correspondant à l'entité association est créée automatiquement. Elle contient des clés primaires qui font référence aux deux autres tables. Ces deux tables ne changent pas contrairement à une relation fonctionnelle. Tandis que dans la version du cours, l'entité association est représentée par une bulle et non une table.

Modèle entités-association



Script SQL généré par l'AGL

```sql
-- MySQL Script generated by MySQL Workbench
-- Fri Jan 13 17:01:29 2023
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DAT

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.``
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`` ;
```

```sql
DROP TABLE IF EXISTS `mydb`.`` ;

CREATE TABLE IF NOT EXISTS `mydb`.`` (
)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`PERSONNE`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`PERSONNE` ;

CREATE TABLE IF NOT EXISTS `mydb`.`PERSONNE` (
  `idPers` INT NOT NULL,
  `nom` VARCHAR(45) NULL,
  `prenom` VARCHAR(45) NULL,
  PRIMARY KEY (`idPers`),
  UNIQUE INDEX `idPers_UNIQUE` (`idPers` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ENSEIGNANT`
```

```
-- Table `mydb`.`ENSEIGNANT`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ENSEIGNANT` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ENSEIGNANT` (
  `PERSONNE_idPers` INT NOT NULL,
  PRIMARY KEY (`PERSONNE_idPers`),
  CONSTRAINT `fk_ENSEIGNANT_PERSONNE1`
    FOREIGN KEY (`PERSONNE_idPers`)
    REFERENCES `mydb`.`PERSONNE` (`idPers`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`MODULE`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`MODULE` ;

CREATE TABLE IF NOT EXISTS `mydb`.`MODULE` (
  `idModule` INT NOT NULL,
  `intitulé` VARCHAR(45) NULL,
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`MODULE` (
  `idModule` INT NOT NULL,
  `intitulé` VARCHAR(45) NULL,
  `UE` VARCHAR(45) NULL,
  `ENSEIGNANT_PERSONNE_idPers` INT NULL,
  PRIMARY KEY (`idModule`),
  UNIQUE INDEX `idModule_UNIQUE` (`idModule` ASC) VISIBLE,
  INDEX `fk_MODULE_ENSEIGNANT1_idx` (`ENSEIGNANT_PERSONNE_idPers` ASC) VISIBLE,
  CONSTRAINT `fk_MODULE_ENSEIGNANT1`
    FOREIGN KEY (`ENSEIGNANT_PERSONNE_idPers`)
    REFERENCES `mydb`.`ENSEIGNANT` (`PERSONNE_idPers`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ÉTUDIANT`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ÉTUDIANT` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ÉTUDIANT` (
  `NIP` VARCHAR(45) NULL,
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`ÉTUDIANT` (
  `NIP` VARCHAR(45) NULL,
  `PERSONNE_idPers` INT NOT NULL,
  PRIMARY KEY (`PERSONNE_idPers`),
  CONSTRAINT `fk_ÉTUDIANT_PERSONNE1`
    FOREIGN KEY (`PERSONNE_idPers`)
    REFERENCES `mydb`.`PERSONNE` (`idPers`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ÉTUDIANT`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ÉTUDIANT` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ÉTUDIANT` (
  `NIP` VARCHAR(45) NULL,
  `PERSONNE_idPers` INT NOT NULL,
  PRIMARY KEY (`PERSONNE_idPers`),
  CONSTRAINT `fk_ÉTUDIANT_PERSONNE1`
    FOREIGN KEY (`PERSONNE_idPers`)
```

```
    FOREIGN KEY (`PERSONNE_idPers`)
    REFERENCES `mydb`.`PERSONNE` (`idPers`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ÉVALUATION`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ÉVALUATION` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ÉVALUATION` (
  `idÉval` INT NOT NULL,
  `nomÉval` VARCHAR(45) NULL,
  `dateÉval` DATE NULL,
  `MODULE_idModule` INT NULL,
  PRIMARY KEY (`idÉval`),
  UNIQUE INDEX `idÉval_UNIQUE` (`idÉval` ASC) VISIBLE,
  INDEX `fk_ÉVALUATION_MODULE1_idx` (`MODULE_idModule` ASC) VISIBLE,
  CONSTRAINT `fk_ÉVALUATION_MODULE1`
    FOREIGN KEY (`MODULE_idModule`)
    REFERENCES `mydb`.`MODULE` (`idModule`)
```

```
    REFERENCES `mydb`.`MODULE` (`idModule`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ÉVALUATION_has_MODULE`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ÉVALUATION_has_MODULE` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ÉVALUATION_has_MODULE` (
  `ÉVALUATION_idÉval` INT NOT NULL,
  `MODULE_idModule` INT NOT NULL,
  PRIMARY KEY (`ÉVALUATION_idÉval`, `MODULE_idModule`),
  INDEX `fk_ÉVALUATION_has_MODULE_MODULE1_idx` (`MODULE_idModule` ASC) VISIBLE,
  INDEX `fk_ÉVALUATION_has_MODULE_ÉVALUATION1_idx` (`ÉVALUATION_idÉval` ASC) VISIBLE,
  CONSTRAINT `fk_ÉVALUATION_has_MODULE_ÉVALUATION1`
    FOREIGN KEY (`ÉVALUATION_idÉval`)
    REFERENCES `mydb`.`ÉVALUATION` (`idÉval`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_ÉVALUATION_has_MODULE_MODULE1`
```

```
  CONSTRAINT `fk_ÉVALUATION_has_MODULE_MODULE1`
    FOREIGN KEY (`MODULE_idModule`)
    REFERENCES `mydb`.`MODULE` (`idModule`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ÉTUDIANT_has_ÉVALUATION`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ÉTUDIANT_has_ÉVALUATION` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ÉTUDIANT_has_ÉVALUATION` (
  `ÉTUDIANT_PERSONNE_idPers` INT NULL,
  `ÉVALUATION_idÉval` INT NULL,
  `note` VARCHAR(45) NULL,
  PRIMARY KEY (`ÉTUDIANT_PERSONNE_idPers`, `ÉVALUATION_idÉval`),
  INDEX `fk_ÉTUDIANT_has_ÉVALUATION_ÉVALUATION1_idx` (`ÉVALUATION_idÉval` ASC) VISIBLE,
  INDEX `fk_ÉTUDIANT_has_ÉVALUATION_ÉTUDIANT1_idx` (`ÉTUDIANT_PERSONNE_idPers` ASC) VISIBLE,
  CONSTRAINT `fk_ÉTUDIANT_has_ÉVALUATION_ÉTUDIANT1`
    FOREIGN KEY (`ÉTUDIANT_PERSONNE_idPers`)
    REFERENCES `mydb`.`ÉTUDIANT` (`PERSONNE_idPers`)
```

```
174       REFERENCES `mydb`.`ÉTUDIANT` (`PERSONNE_idPers`)
175       ON DELETE NO ACTION
176       ON UPDATE NO ACTION,
177     CONSTRAINT `fk_ÉTUDIANT_has_ÉVALUATION_ÉVALUATION1`
178       FOREIGN KEY (`ÉVALUATION_idÉval`)
179       REFERENCES `mydb`.`ÉVALUATION` (`idÉval`)
180       ON DELETE NO ACTION
181       ON UPDATE NO ACTION)
182   ENGINE = InnoDB;
183
184
185   -- -----------------------------------------------------
186   -- Table `mydb`.`ENSEIGNANT`
187   -- -----------------------------------------------------
188   DROP TABLE IF EXISTS `mydb`.`ENSEIGNANT` ;
189
190   CREATE TABLE IF NOT EXISTS `mydb`.`ENSEIGNANT` (
191     `PERSONNE_idPers` INT NOT NULL,
192     PRIMARY KEY (`PERSONNE_idPers`),
193     CONSTRAINT `fk_ENSEIGNANT_PERSONNE1`
194       FOREIGN KEY (`PERSONNE_idPers`)
195       REFERENCES `mydb`.`PERSONNE` (`idPers`)
196       ON DELETE NO ACTION
```

```
196       ON DELETE NO ACTION
197       ON UPDATE NO ACTION)
198   ENGINE = InnoDB;
199
200
201   -- -----------------------------------------------------
202   -- Table `mydb`.`Participe`
203   -- -----------------------------------------------------
204   DROP TABLE IF EXISTS `mydb`.`Participe` ;
205
206   CREATE TABLE IF NOT EXISTS `mydb`.`Participe` (
207     `ÉTUDIANT_PERSONNE_idPers` INT NOT NULL,
208     `MODULE_idModule` INT NOT NULL,
209     PRIMARY KEY (`ÉTUDIANT_PERSONNE_idPers`, `MODULE_idModule`),
210     INDEX `fk_ÉTUDIANT_has_MODULE_MODULE1_idx` (`MODULE_idModule` ASC) VISIBLE,
211     INDEX `fk_ÉTUDIANT_has_MODULE_ÉTUDIANT1_idx` (`ÉTUDIANT_PERSONNE_idPers` ASC) VISIBLE,
212     CONSTRAINT `fk_ÉTUDIANT_has_MODULE_ÉTUDIANT1`
213       FOREIGN KEY (`ÉTUDIANT_PERSONNE_idPers`)
214       REFERENCES `mydb`.`ÉTUDIANT` (`PERSONNE_idPers`)
215       ON DELETE NO ACTION
216       ON UPDATE NO ACTION,
217     CONSTRAINT `fk_ÉTUDIANT_has_MODULE_MODULE1`
218       FOREIGN KEY (`MODULE_idModule`)
```

```
206  CREATE TABLE IF NOT EXISTS `mydb`.`Participe` (
207    `ÉTUDIANT_PERSONNE_idPers` INT NOT NULL,
208    `MODULE_idModule` INT NOT NULL,
209    PRIMARY KEY (`ÉTUDIANT_PERSONNE_idPers`, `MODULE_idModule`),
210    INDEX `fk_ÉTUDIANT_has_MODULE_MODULE1_idx` (`MODULE_idModule` ASC) VISIBLE,
211    INDEX `fk_ÉTUDIANT_has_MODULE_ÉTUDIANT1_idx` (`ÉTUDIANT_PERSONNE_idPers` ASC) VISIBLE,
212    CONSTRAINT `fk_ÉTUDIANT_has_MODULE_ÉTUDIANT1`
213      FOREIGN KEY (`ÉTUDIANT_PERSONNE_idPers`)
214      REFERENCES `mydb`.`ÉTUDIANT` (`PERSONNE_idPers`)
215      ON DELETE NO ACTION
216      ON UPDATE NO ACTION,
217    CONSTRAINT `fk_ÉTUDIANT_has_MODULE_MODULE1`
218      FOREIGN KEY (`MODULE_idModule`)
219      REFERENCES `mydb`.`MODULE` (`idModule`)
220      ON DELETE NO ACTION
221      ON UPDATE NO ACTION)
222  ENGINE = InnoDB;
223
224
225  SET SQL_MODE=@OLD_SQL_MODE;
226  SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
227  SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
228
```

Les différences entre les scripts manuel/AGL

Premièrement, on peut apercevoir que l'AGL utilise les guillemet-apostrophe pour nommer les noms des tables ce qu'on ne fait pas forcément lorsqu'on écrit des scripts manuellement sur Postgresql.

De plus, des commentaires sont écrits pour indiquer chaque création de table ce qui est un bon point pour se repérer dans un long script. Or, manuellement on ne le fait pas forcément.

Un schéma est créé automatiquement, je pense que celui-ci correspond au fichier mydb. De plus, une table mydb est également créée. On peut remarquer que chaque table est nommée ainsi : 'mydb.nom_table'. Ceci indique peut-être qu'il s'agit d'une table qui se situe dans le fichier mydb, comme une bibliothèque. Tout ceci n'est pas pris en compte dans le script manuel.

Les index qu'on retrouvait dans la modélisation de l'AGL sont également créés automatiquement tandis que dans un script écrit manuellement les index ne sont pas indiqués.

Dans le script de création d'une table, l'ordre des informations est le même qu'un script écrit manuellement dans Postgresql. Il est indiqué le type et les contraintes. Pour indiquer les contraintes de colonne le mot clé CONTRAINT est utilisé suivi du nom de la colonne et les contraintes de celle-ci. Dans les scripts écrits en cours on ne précise pas le mot CONTRAINT.

Enfin, à chaque fin de création de table il est écrit "Engine = InnoDB" pour indiquer le moteur de stockage pour contrôler, lire et enregistrer les informations dans les bases de données. Ici il s'agit de InnoDB. Cette indication n'est pas faite dans le script manuel.

3. Peuplement des tables et requêtes

Étapes du peuplement

Création de la table DATA :

```
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 VARCHAR,

   note VARCHAR,

   id_etudiant INTEGER,

   nom_etudiant VARCHAR,
```

```
   prenom_etudiant VARCHAR);
```

J'ai créé une table data dont les colonnes correspondent aux colonnes du tableau se trouvant dans le fichier data.csv. Je n'ai mis aucune contrainte de colonne, car la table sert seulement à remplir les autres tables.

Insertion des valeurs du fichier data.csv dans la table data :

```
\copy data FROM 'Documents/SAE/SAE_1.03/data.csv' WITH (FORMAT CSV, HEADER, DELIMITER
';');
```

Ici, j'ai copié dans le fichier data.csv les valeurs en précisant le format et le délimiteur afin d'indiquer à postgresql quand est-ce-qu'il faut passer à la colonne suivante. De plus, le HEADER est là pour spécifier que la première ligne du tableau dans le fichier ne compte pas.

Étape 1 : Insertion des valeurs dans les tables indépendantes

Insertion des valeurs dans la table personne à partir de la table data :

```
INSERT INTO

   personne (id_personne, nom_personne, prenom_personne)

SELECT

   DISTINCT id_enseignant,

   nom_enseignant,

   prenom_enseignant

FROM

   data;



INSERT INTO

   personne (id_personne, nom_personne, prenom_personne)

SELECT
```

```
   DISTINCT id_etudiant,
   nom_etudiant,
   prenom_etudiant
FROM
   data;
```

Insertion des valeurs dans la table enseignant à partir de la table data :

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

Insertion des valeurs dans la table etudiant à partir de la table data :

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

Étape 2 : Insertion des valeurs dans les tables dépendantes

Insertion des valeurs dans la table module à partir de la table data :

```
INSERT INTO

   module (id_module, intitule, code, ue, id_enseignant)

SELECT

   DISTINCT id_module,

   intitule_module,

   code,

   ue,

   id_enseignant

FROM

   data;
```

Insertion des valeurs dans la table evaluation à partir de la table data :

```
INSERT INTO

   evaluation (nom_eval, date_eval, id_module)

SELECT

   DISTINCT nom_evaluation,

   date_evaluation,

   id_module

FROM

   data;
```

Insertion des valeurs dans la table participe à partir de la table data :

J'ai inséré le id_etudiant et id_module dans la table data pour avoir tous les modules qui sont dans la même ligne que l'étudiant.

```
INSERT INTO

   participe (id_module, id_etudiant)

SELECT

   DISTINCT id_module,

   id_etudiant

FROM

   data;
```

Insertion des valeurs dans la table aFait à partir de la table data :

J'ai fait une jointure entre la table evaluation et data afin de pouvoir insérer les valeurs de id_eval des évaluations que l'étudiant a effectué dans la table aFait ainsi que les notes associées à cette évaluation.

```
INSERT INTO

   aFait (id_eval, id_etudiant, note)

SELECT

   id_eval,

   id_etudiant,

   note

from

   data

   join evaluation on data.nom_evaluation = evaluation.nom_eval

   and data.date_evaluation = evaluation.date_eval;
```

Présentation de 2 requêtes sur la base de données

Requête sur le nombre d'évaluation effectué :

Cette requête permet de voir quels sont les élèves qui n'ont pas participé à toutes les évaluations.

```
SELECT

   nom_personne,

   prenom_personne,

   COUNT(*)

FROM

   aFait

   JOIN personne ON id_etudiant = id_personne

GROUP BY

   nom_personne,

   prenom_personne

ORDER BY

   nom_personne;



SELECT

   nom_evaluation,

   nom_etudiant,

   prenom_etudiant

FROM

  data WHERE nom_etudiant = 'Point' and prenom_etudiant = 'Yvon';
```

Requête sur les moyennes des élèves :

Avant de faire cette requête j'ai modifié le type de l'attribut "note" dans la table aFait car celle-ci était un VARCHAR pour l'insertion des données du fichier data.csv.

```
ALTER TABLE

   afait

ALTER COLUMN

   note TYPE FLOAT USING (note :: double precision);
```

Cette requête permet de filtrer les élèves en fonction de leur moyenne. Je sélectionne les élèves qui ont une moyenne supérieure ou égale à 10.

```
SELECT

   nom_personne,

   prenom_personne,

   AVG(note)

FROM

   aFait

   JOIN personne ON id_etudiant = id_personne

GROUP BY

   nom_personne,

   prenom_personne

HAVING

   AVG(note) >= 10;
```