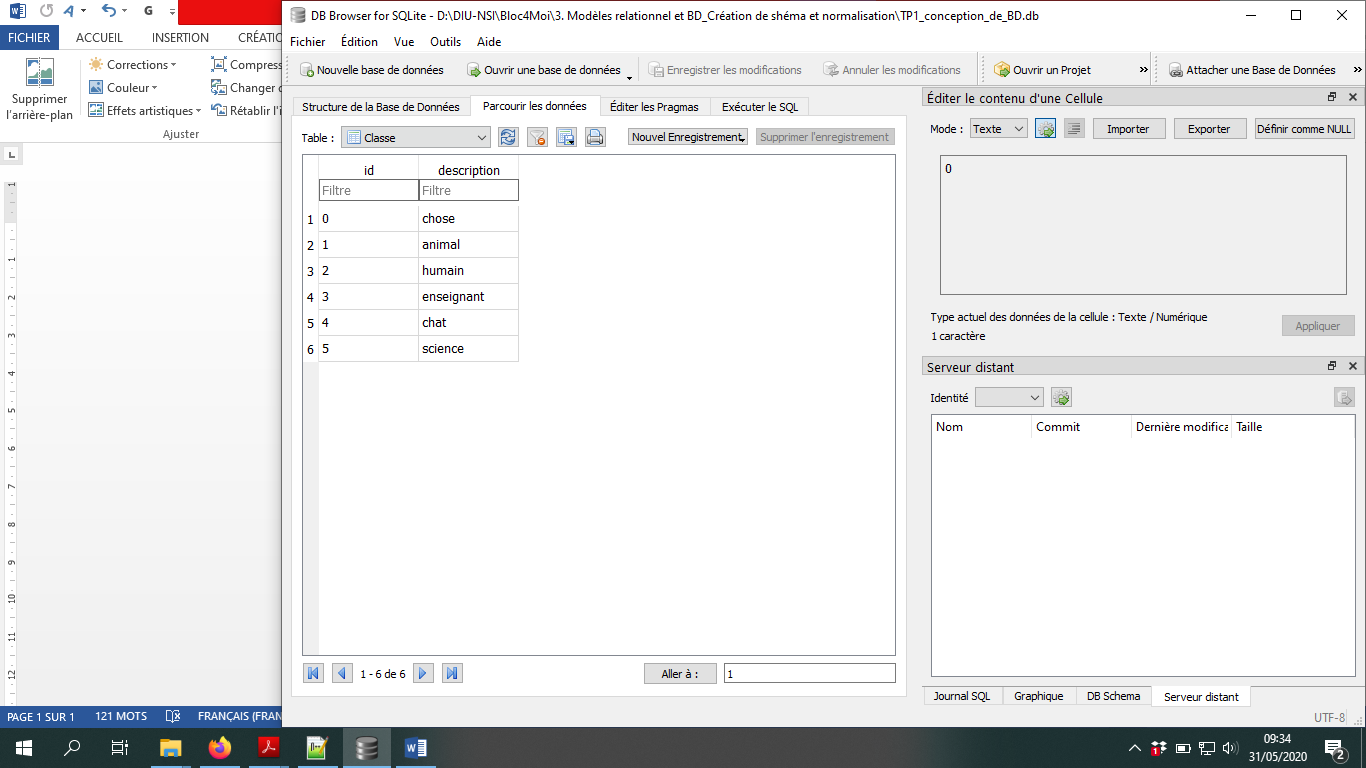
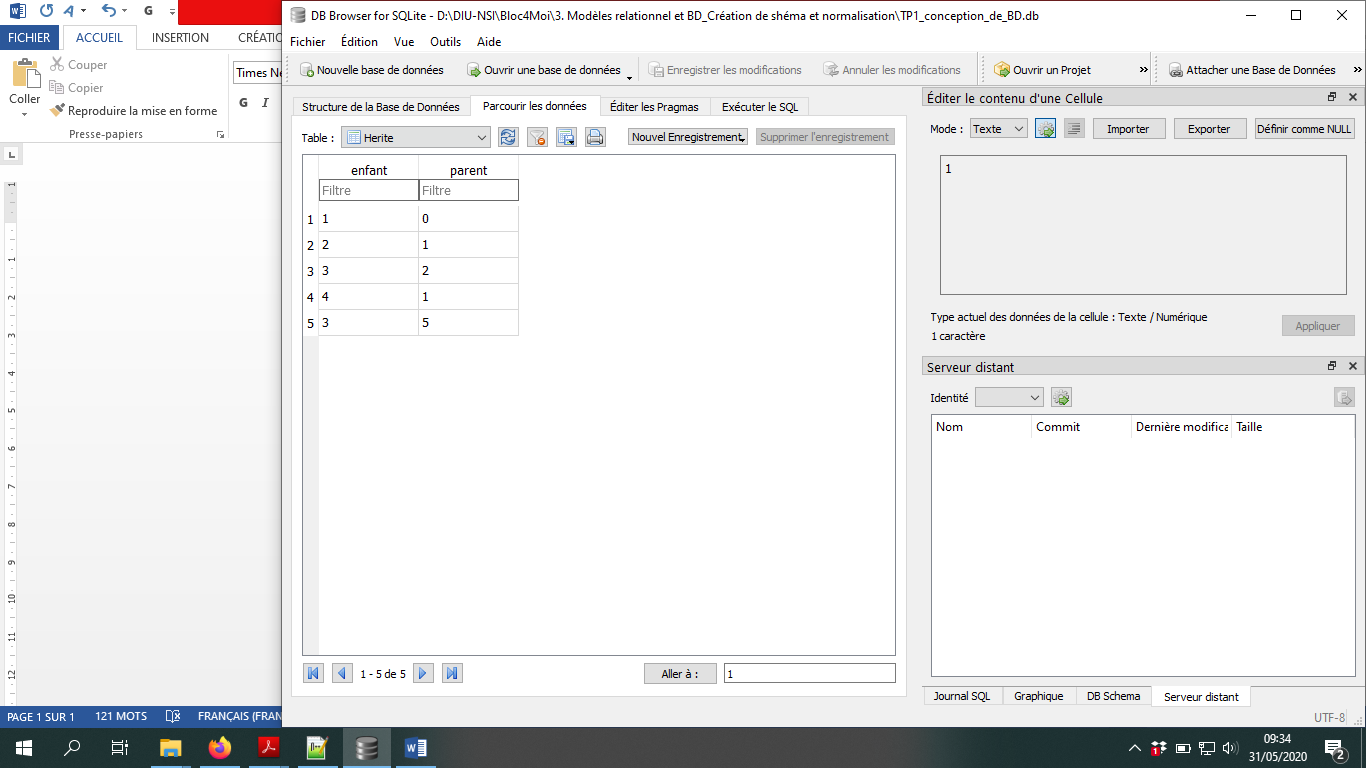
**TP1\_conception\_de\_BD**

Exercice : variations sur le thème de la clef primaire



PRAGMA foreign\_keys=1;

CREATE TABLE Classe(

id INTEGER PRIMARY KEY,

description TEXT

);

CREATE TABLE Herite(

enfant INT REFERENCES Classe(id),

parent INT REFERENCES Classe(id),

PRIMARY KEY (enfant, parent)

);

**\*\*Exercice\*\* : on veut s'habituer aux messages d'erreurs standards :**

\* insérez un nouveau tuple avec `INSERT INTO Classe VALUES(0,'patate'); Quel est le message d'erreur ?

Result: UNIQUE constraint failed: Classe.id

At line 25:

INSERT INTO Classe VALUES(0, 'patate');

En effet, dans Classe, id est clé primaire donc on ne peut pas avoir deux lignes différentes avec même clé primaire et comme à 0 on a déjà affecté chose, on ne peut pas insérer (0, 'patate'); On a violé l’unicité.

\* Idem avec `INSERT INTO Herite VALUES(1, 0);`

Result: UNIQUE constraint failed: Herite.enfant, Herite.parent

At line 25:

INSERT INTO Herite VALUES(1, 0);

En effet, dans Herite, (enf,par) est clé primaire donc on ne peut pas avoir deux lignes différentes avec même clé primaire et comme à (1,0) est déjà, on ne peut pas l’insérer à nouveau;

\* insérez un nouveau tuple avec `INSERT INTO Herite VALUES(0, 42);`. Quel est le message d'erreur ?

(\_NB\_ : si vous n'en avez pas, c'est que les clefs étrangères ne sont pas activées !)

Result: FOREIGN KEY constraint failed

At line 25:

INSERT INTO Herite VALUES(0, 42);

42 doit pointer sur une clé de Classe… c'est clair, mais moins descriptif (il pourrait y avoir plusieurs FK)

Remarque : on peut insérer (0,1) ! INSERT INTO Herite VALUES(0, 1);

\*\*Exercice\*\* : dessinez un diagramme Entité-Association du schéma de cette base.

Ensuite, représentez graphiquement \_le contenu\_ de cette base par une hiérarchie entre les classes.

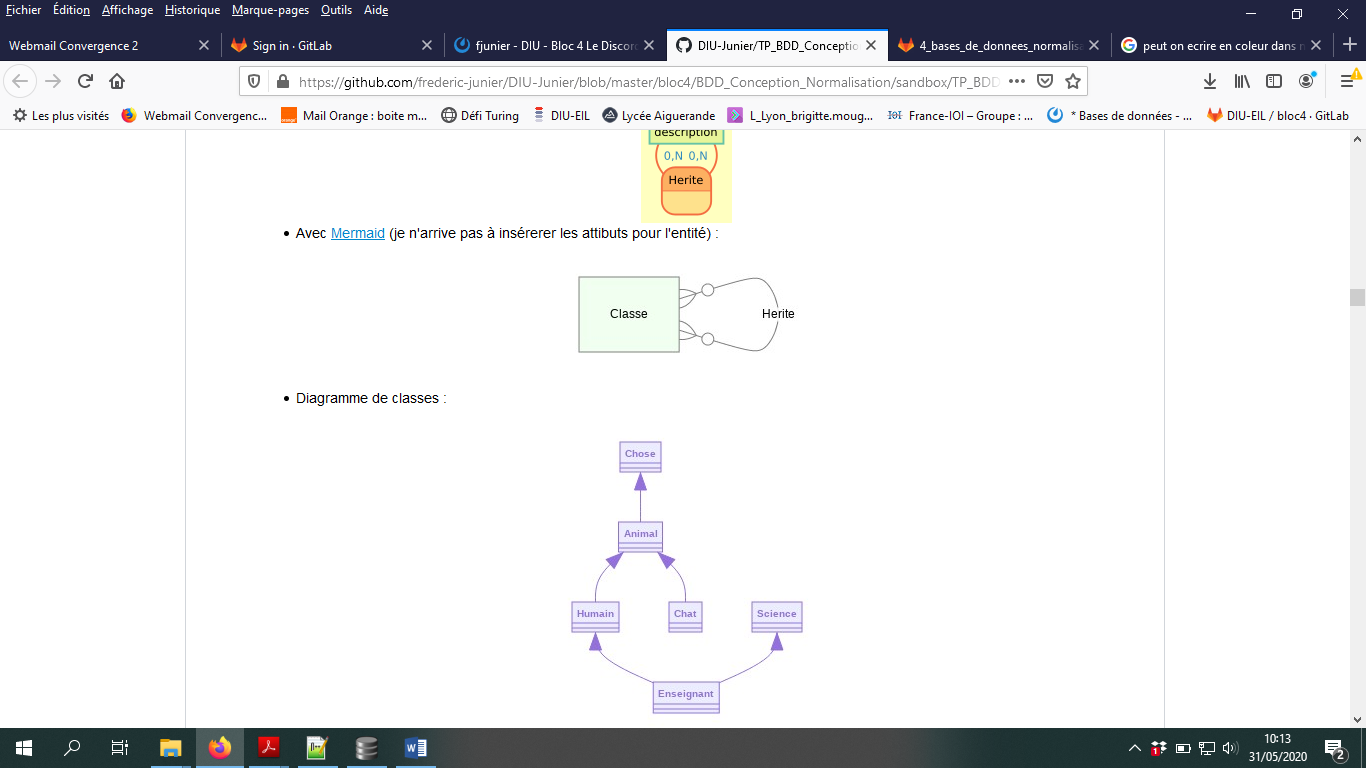
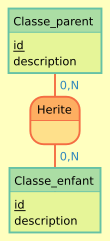
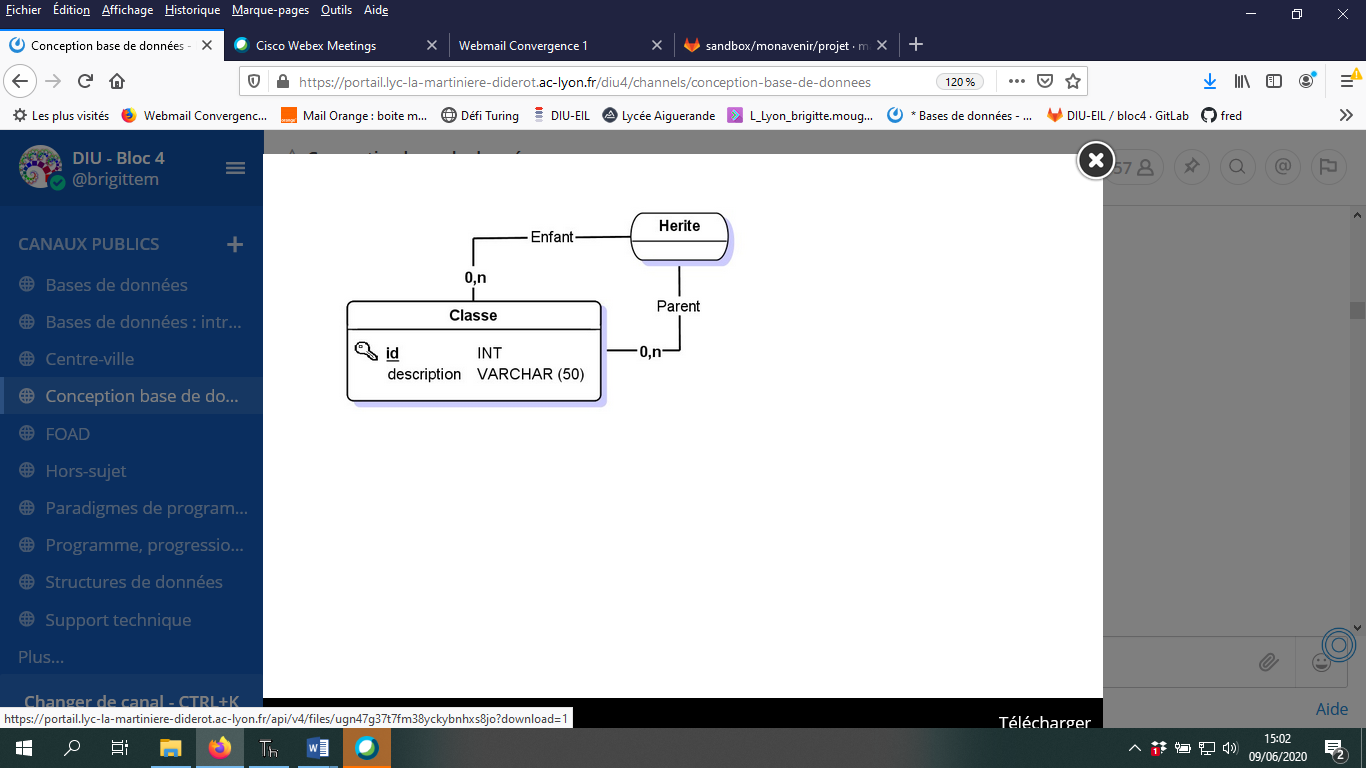
● Premier diagramme : ● Avec une entité- ● Hiérarchie entre classes :  
association réflexive :

Diagramme e/a attendu :



**Commentaires Thion :**

Voir le fichier JMerise `TP\_conception\_de\_BD\_EA.mcd` (je n’arrive pas à l’ouvrir…).

Pour la hiérarchie, on remarque surtout qu’on peut avoir 2 fils (`animal`) ou 2 pères (`enseignant`) et donc que cette hiérarchie ce n'est pas un arbre mais par contre elle est sans cycle ! Jouer avec la génération de SQL de JMerise (dans "Scripts & BD", "Scripts SQL" puis "SQlite") : obtenir le schéma de départ au renommage près.

**\*\*Exercice\*\* : si on remplace `PRIMARY KEY (enfant, parent)` par la contrainte `PRIMARY KEY (enfant)` dans la définition de `Herite` :**

\* quelle restriction est imposée sur les hiérarchies de classes que cette base peut représenter ?

On ne peut pas avoir 2 p-uplets avec la même clé primaire enfant : donc on ne pourra pas insérer dans la table Herite deux relations d'héritage avec le même identifiant enfant et des identifiants parent distincts. (3,2) impose qu’on ne peut plus mettre (3,5)… Autrement dit, **un enfant ne peut avoir qu'un parent**, une classe enfant ne peut hériter que d'une seule classe mère.

\* quel est le changement correspondant dans le diagramme E/A ?

Changement de cardinalité sur l'une des pattes (sur un des arcs) de l'association Herite, le faire dans JMErise…

On met 0,1 sur la patte à partir de enfant… entouré en rouge sur diagramme du dessus…

\* si on retraduit le diagramme E/A en schéma SQL, quel schéma obtiendra-t-on ?

SOLUTION 1 : On garde CREATE CLASSE idem… SOLUTION 2 :

CREATE TABLE Herite( CREATE TABLE Classe(

enfant INT REFERENCES Classe(id), id INTEGER PRIMARY KEY,

parent INT REFERENCES Classe(id), description TEXT

PRIMARY KEY enfant ); seul chgmt parent INT REFERENCES Classe(id));

SOL 1 : "enfant" est clé étrangère car elle fait le lien entre les 2 tables et elle est clé primaire car elle identifie un tuple de façon unique dans herite (dans une asso, c'est fréquent d’avoir une clé primaire qui est étrangère)

SOL 2 : on n’obtient plus qu'une seule table avec un attribut `parent` : vu que chaque classe ne peut avoir qu'au plus un seul parent, pas la peine de faire une relation binaire `Herite` pour ça (un seul attr dans la classe suffit).

**\*\*Exercice\*\* : mêmes questions que précédemment si on remplace `PRIMARY KEY (enfant, parent)` par la contrainte `PRIMARY KEY (parent)` .**

Symétrique à question pcdte !

Un **parent ne peut avoir qu'un enfant**. Changement de cardinalité sur l'une des pattes de l'association Herite,

On met 0,1 sur la patte à partir de parent… entouré en vert sur diagramme du dessus…

Et pareil, deux représentations possibles…

**\*\*Exercice\*\* : mêmes questions que précédemment si on enlève simplement la contrainte `PRIMARY KEY (enfant, parent)`.**

On a toujours une contrainte d'intégrité avec la clef id dans la table Classe, on ne peut pas insérer un tuple avec un id déjà attribué, comme le INSERT INTO Classe VALUES(0, 'patate');

En revanche, redondances possibles : on n'a plus de contrainte d'intégrité sur l'unicité dans la table Herite (qui n'a plus de clef primaire). Donc INSERT INTO Herite VALUES(1, 0); fonctionne…

On a toujours la contrainte d'intégrité référentielle de la colonne id de la table Classe. On ne peut pas insérer si l’identifiant n’est pas dans la table Classe : INSERT INTO Herite VALUES(0, 42); ne va pas.

Autrement dit : **on peut être plusieurs fois l'enfant d'un de ses parents**

* Diagramme entité relation dans ce cas :

Ici il y a un piège car c’est pas modélisable, **on n’est plus en model relationnel**, pas de cahier des charges…

Pas traduisible en schéma e/a si plus de clé primaire. Penser : **y’a toujours une clé primaire** !

***Remarques générales* :** Quand on est identifiable par ses attributs on est une entité et quand les clés étrangères font partie de la clé primaire, on est une association.

Quand la clé primaire est composée de clés étrangères alors typiquement on est défini par les entités qui participent, on est une association on n’est pas une entité forte !

L'entité "Classe" est en association avec elle-même. Si b détermine c, alors b est clé.

Quand les problèmes arrivent tardivement dans une application, c’est typiquement un problème de bd.

La bd doit correspondre au cahier des charges.

Exercice : reprise de la base \_Stanford\_

On reprend la base \_Stanford\_ d'exemple de la première partie. Le script de création de table est le suivant.

**\*\*Exercice\*\* : Reprenez le fichier `base-stanford.sql` du TP1 et modifiez le schéma pour y ajouter les contraintes d'intégrité (clef primaire et clef étrangères) attendues.**

**Après modification, toutes les données d'exemple de la base doivent pouvoir être insérées sans erreurs.**

PRAGMA foreign\_keys=1;

DROP table if exists Apply;

DROP table if exists College;

DROP table if exists Student;

create table College(

cName varchar(255) PRIMARY KEY,

state varchar(255),

enrollment int );

create table Student(

sID int PRIMARY KEY,

sName varchar(255),

GPA real CHECK(0 <= GPA <= 4), -- Grade Point Average, out of 4 ou écrire BETWEEN

sizeHS int );

create table Apply(

sID int REFERENCES Student(sID),

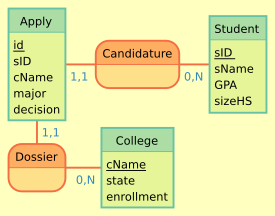
cName varchar(255) REFERENCES College(cName),

major varchar(255),

decision char(1) CHECK(decision IN('Y','N')),

PRIMARY KEY (sID, cName, major));

* Diagramme entité relation dans ce cas (j'ai rajouté une clef primaire unitaire au lieu du triplet sID, cName,major) id dans la table Apply) :

****

**Apply** ( id, sID, cName, major, decision )

* Le champ *id* constitue la clef primaire de la table.
* Les champs *sID*, *cName*, *major* et *decision* sont des attributs.

**Student** ( sID, sName, GPA, sizeHS )

* Le champ *sID* constitue la clef primaire de la table.
* Les champs *sName*, *GPA* et *sizeHS* sont des attributs.

**College** ( cName, state, enrollment )

* Le champ *cName* constitue la clef primaire de la table.
* Les champs *state* et *enrollment* étaient déjà de simples attributs de l'entité *College*.

**\*\*Exercice\*\* : SQL propose un type de contrainte assez générique appelée `CHECK`, définissez en pour les attributs \_GPA\_ et \_decision\_. Là encore, toutes les données doivent pouvoir s'importer correctement après ajout des contraintes supplémentaires.**

(<https://en.wikipedia.org/wiki/Academic_grading_in_the_United_States#Numerical_and_letter_grades>)

insert into Student values (843, 'Craig', -3.4, 2000); ERROR !!

insert into Student values (843, 'Craig', 4.4, 2000); ERROR !!

insert into Apply values (234, 'MIT', 'CS', 'Yes'); ERROR !!

**On remarque que dans cette base, les disciplines sont identifiées par une chaîne de caractères non normalisée (casse variable, abréviations). On va donc créer une table pour les disciplines et utiliser les requêtes d'insertion et de mise-à-jour. Quelques liens de documentation :**

\* `INSERT` <https://www.sqlite.org/lang_insert.html> et <https://www.sqlitetutorial.net/sqlite-insert/>

\* `UPDATE` <https://www.sqlitetutorial.net/sqlite-update/> et <https://www.sqlite.org/lang_update.html>

**\*\*Exercice\*\* : avec une requête, déterminez la valeur de \_major\_ la plus longue. Il faudra un agrégat et trouver la bonne fonction sur les chaînes dans la documentation** <https://www.sqlite.org/lang_corefunc.html>

SELECT MAX(LENGTH(major)),major FROM Apply;

SELECT DISTINCT major FROM Apply

WHERE LENGTH(major) = (SELECT MAX(LENGTH(major)) FROM Apply );

SELECT major FROM Apply

GROUP BY major

HAVING LENGTH(major) = (SELECT MAX(LENGTH(major)) FROM Apply );

**\*\*Exercice\*\* : définir une nouvelle table `Major` avec comme attributs \_code\_ qui est clef et un autre attribut \_description\_. Peuplez ensuite cette table en utilisant l'attribut \_major\_ de `Apply` (avec une requête `INSERT INTO Apply SELECT ...`). Lors de l'import, normalisez l'attribut \_code\_ pour que toutes ses valeurs soient en majuscule. Pour la description, vous pouvez remplir avec le code dans un premier temps puis éditer à la main (e.g., pour \_EE\_ on voudrait lire \_Electrical Engineering\_).**

DROP TABLE IF EXISTS Major;

CREATE TABLE Major(

code varchar(16) PRIMARY KEY, --au moins 14, on prend 16 pour plus de sécurité…

description varchar(255) NOT NULL );

INSERT INTO Major SELECT DISTINCT

UPPER(substr(major, 1, 2)), upper(major) FROM Apply ;

substr(major, 1, 2)) permet d’écrire les deux premiers caractères

**\*\*Exercice\*\* : on souhaite mettre à jour la table `Apply` et ajouter la contrainte de clef étrangère vers la table `Major` nouvellement créée. Malheureusement, SQLite ne permet pas les opérations nécessaires.** Voir <https://www.sqlite.org/omitted.html> Voir aussi <https://www.sqlitetutorial.net/sqlite-check-constraint/>

**Ainsi à la place :**

. créez une table `Apply2` avec toutes les contraintes,

. insérez le contenu de `Apply` dans `Apply2`,

. supprimez `Apply` (commande `DROP TABLE ...`),

. enfin renommez `Apply2` en `Apply` (commande `ALTER TABLE ....`).

CREATE TABLE Apply2(

sID int REFERENCES Student(sID),

cName varchar(255) REFERENCES College(cName),

codeMajor varchar(255) REFERENCES Major(code),

decision char(1) CHECK(decision IN('Y','N')),

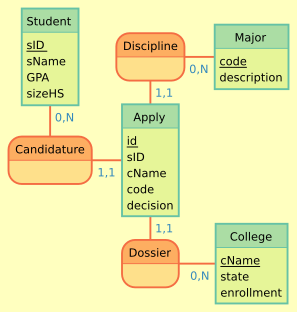
PRIMARY KEY (sID, cName, codeMajor));

INSERT INTO Apply2 SELECT sID, cName, upper(major), decision FROM Apply ;

DROP TABLE Apply;

ALTER TABLE Apply2 RENAME TO Apply;

* Diagramme entité relation dans ce cas :



**Student** ( sID, sName, GPA, sizeHS )

* Le champ *sID* constitue la clef primaire de la table.
* Les champs *sName*, *GPA* et *sizeHS* sont des attributs

**Major** ( code, description )

* Le champ *code* constitue la clef primaire de la table.
* Le champ *description* est un attribut de l'entité *Major*.

**Apply** ( id, sID, cName, code, decision)

* Le champ *id* constitue la clef primaire de la table.
* Les champs *sID*, *cName*, *code* et *decision* sont des attributs.

**College** ( cName, state, enrollment )

* Le champ *cName* constitue la clef primaire de la table.
* Les champs *state* et *enrollment* sont attributs de l'entité *College*.

TP : CaveVins

<http://mocodo.wingi.net/>

Stocké à, 0N Site, 11 Stockage

A été dégustée, 11 Dégustation, 0N Stockage

Admet pour stockage, 1N LotBouteille, 11 Stockage

De la cuvée, 01 Domaine, 0N Cuvée

Du domaine, 0N Domaine, 11 LotBouteille

Dans la région, 11 Domaine, 0N Région

Commentaires lors de l'achat, 01 LotBouteille, 11 Commentaires

Site: idSite, lieuStockage, précision

Stockage: idStockage, idLotBouteille, idSite, nombre

LotBouteille: idLotBouteille, idCommentaires, idDomaine, couleur, volume, millésime, degré

Dégustation: idDégustation, idStockage, dateDégustation, commentairesDégustation

Région: idRégion, nomRégion, appellation

Domaine: idDomaine, idRegion, idCuvée, nomDomaine

Cuvée: idCuvée, nomCuvée

Commentaires: idCommentaires, prix, descriptionDégustation

