

# Projet: A propos de la Gestion des Jeux Olympiques

Duy Minh LE, 12315392

## 1. Conception UML et Normalisation

### Question 1

On rappelle le schéma relationnel fourni:

**LesEpreuves** (numEp, nomEp, formeEp, categorieEp, nbSportifsEp, dateEp, nomDi)

*/\* <no, n, f, c, nb, da, di> ∈ LesEpreuves ⇔ no est le numéro d'épreuve du nom n, forme (individuelle, par equipe ou par couple), catégorie c (féminin, masculin ou mixte), un nombre de sportifs nb et une date d. L'épreuve fait partie de la discipline di \*/*

**LesSportifsEQ** (numSp, nomSp, prenomSp, pays, categorieSp, dateNaisSp, numEq)

*/\* <no, n, p, pa, d, c, ne> ∈ LesSportifs ⇔ no est le numéro de sportif, avec un nom n, un prénom p, un pays pa une date de naissance d et une catégorie c (feminin ou masculin). Il est inscrit dans l'équipe ne. \*/*

On peut en suite en deduire les dependances fonctionnelles

- **LesEpreuves** (numEp, nomEp, formeEp, categorieEp, nbSportifsEp, dateEp, nomDi)

**numEp** → nomEp, formeEP, categorieEP, nbSportifsEp, dateEp, nomDi

On voit bien que **nomEp** determine tous les autres attributs, donc une clé. C'est en fait, la seule clé, on détermine bien que cette relation est de forme **BCNF**.

- **LesSportifsEQ** (numSp, nomSp, prenomSp, pays, categorieSp, dateNaisSp, numEq)

**numSp** → nomSp, prenomSp, pays, categorieSp, dateNaisSp

**numEq** → pays

Ici on a **numEq** qui ne determine que pays, donc via la regle d'augmentation, (**numSp, numEq**) est une clé. Vu que tous les attributs sauf numEq sont dependants a numSp, cette relation est de forme **1NF**.

*Ici le numero d'équipe numEq est dans la clé, bien qu'il y aie des sportifs sans équipe. On va ultimativement attaquer ce problème en transformant le schéma*

On transforme donc la relation **LesSportifsEQ** en la coupant par 2:

- **LesSportifs** (numSp, nomSp, prenomSp, pays, categorieSp, dateNaisSp)

**numSp** → nomSp, prenomSp, pays, categorieSp, dateNaisSp

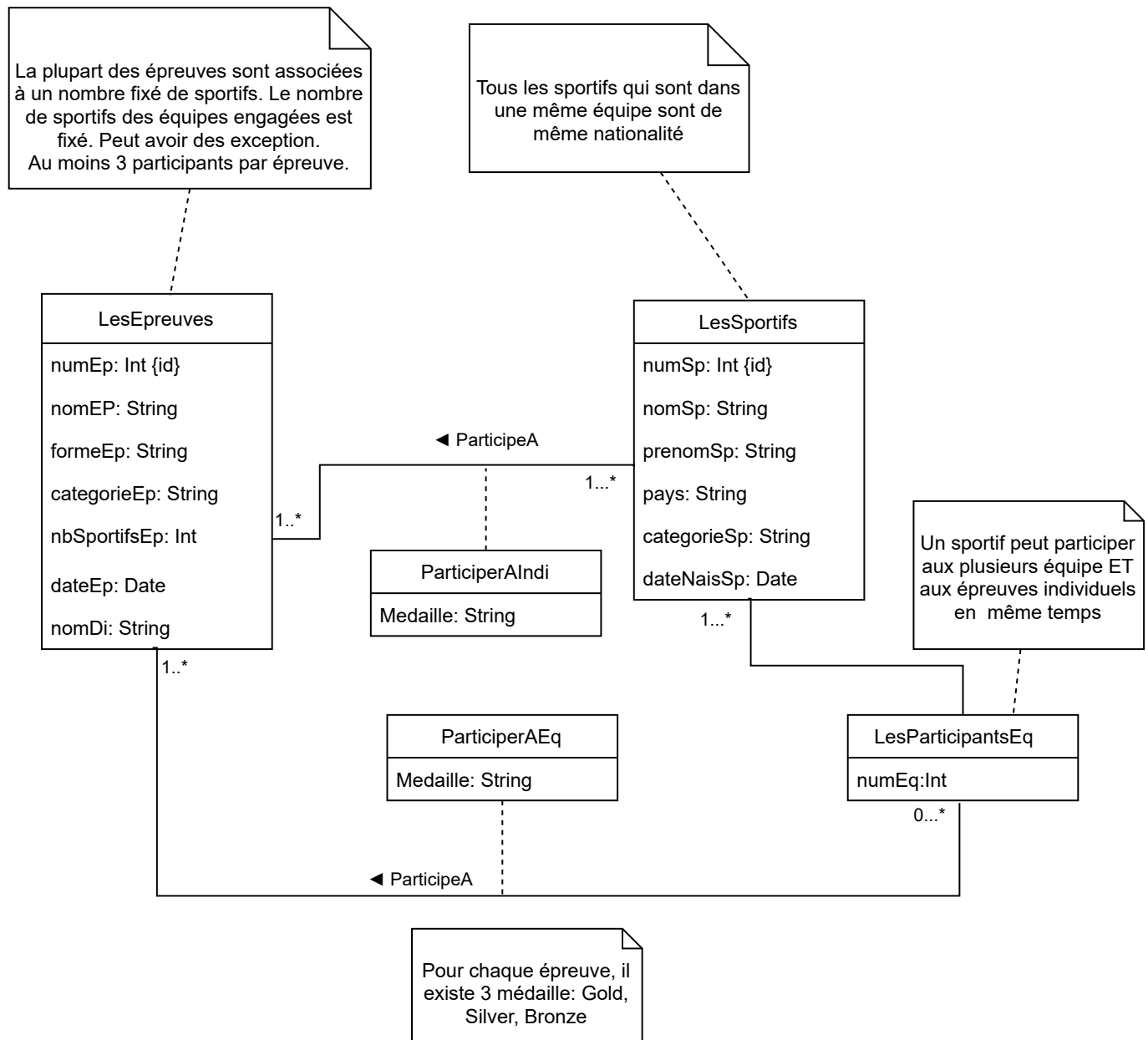
**numSp** est facilement la clé dans ce cas, et cette relation est de forme **BCNF**.

- **LesParticipants** (numSp, numEq)

Pas de DF donc **BCNF** .

## Question 2

On va maintenant modifier le schema obtenu a la question precedente pour completer la BD.



## 2. Implementation

### Question 3

#### Schema propose

- **LesEpreuves** (numEp, nomEp, formeEp, categorieEp, nbSportifsEp, dateEp, nomDi)
- **LesSportifs** (numSp, nomSp, prenomSp, pays, categorieSp, dateNaisSp)
- **LesParticipantsEq** (numEq)
- **RepartitionEq** (numSp, numEq)

- **ParticiperAIndi**(numSp, numEp, medaille)
- **ParticiperAEq**(numEq, numEp, medaille)

## Domaines

- domaine (dateNaisSp) = date(dateEp) = Date
- domaine (formeEp) = {'individuelle', 'par equipe', 'par couple'}
- domaine (categorieEp) = {'feminin', 'masculin', 'mixte'}
- domaine (categorieSp) = {'feminin', 'masculin'}
- domaine (medaille) = {'Gold', 'Silver', 'Bronze'}
- domaine (nomDi) = domaine (nomEp) = domaine (nomSp) = domaine (prenomSp) = domaine (pays) = chaines de caracteres
- domaine (numSp) = domaine (numEp) = domaine (nbSportifsEp) = entier > 0

## Contraintes d'integralite

- $\text{RepartitionEq}[\text{numSp}] \subseteq \text{LesSportifs}[\text{numSp}]$
- $\text{RepartitionEq}[\text{numEq}] \subseteq \text{LesParticipantsEq}[\text{numEq}]$
- $\text{ParticiperAIndi}[\text{numSp}] = \text{LesSportifs}[\text{numSp}]$
- $\text{ParticiperAIndi}[\text{numEp}] \subseteq \text{LesEpreuves}[\text{numEp}]$
- $\text{ParticiperAEq}[\text{numEq}] = \text{LesParticipantsEq}[\text{numEq}]$
- $\text{ParticiperAEq}[\text{numEp}] \subseteq \text{LesEpreuves}[\text{numEp}]$
- $\text{ParticiperAEq}[\text{numEp}] \cap \text{ParticiperAIndi}[\text{numEp}] = \emptyset$

## Question 4

### Etape 2

Je vous laisse regarder le script SQL fourni avec ce rendu.

*Commentaire:* J'ai aussi ajoute les triggers **ON DELETE** aux certains atributs, j'expliquerai dans la partie des triggers.

### Etape 3

De meme, le code python est fourni avec le rendu donc je fais un petit resume des lignes que j'ai modifie/ajoute:

- **main.py**: Ajoute des options 5 et 6 pour reinitialiser les views et les triggers. En fait, dans le script SQL avant de creer un tel view ou trigger, je fais **DROP IF EXISTS**
- **database\_functions.py**: Ajoute les fonctions correspondantes aux options 5 et 6 (pricipalement copier et coller)
- **excel\_extractor.py**: change presque tout le fichier sauf la partie des epreuves. En generalment il n'y a rien qui est complique. Pour l'insertion des medailles, en fait, tout d'abord j'insere tous les sportifs et leurs epreuves dans **ParticiperAIndi** et **ParticiperAEq** avec **'null'** comme medaille. Puis je fais **Update** pour rajouter les propres medailles. Ce qui n'est pas tres efficace, je reconnais bien, mais l'echelle avec laquelle on travaille n'est pas enorme, donc ca suffit.

### Etape 4

Consultez le script `viewDB.sql` fourni.

Pour les triggers, je propose 2 triggers:

- `disqualificationSp`: dès la disqualification d'un sportif, toutes les équipes auxquelles il fait partie sont aussi disqualifiées. Ceci explique bien pourquoi j'ai mis des `ON DELETE CASCADE` sur les foreign keys lors de la création des tables: Une fois un sportif supprimé, on supprime aussi ses inscriptions individuelles, et une fois une équipe disqualifiée, on supprime ses inscriptions aussi. Ce qui est difficile à régler, est l'amélioration des médailles. Vu qu'on n'a pas des informations sur les classements exacts des participants, je garde les médailles intactes.
- `disqualificationEp`: encore une fois avec la disqualification. Si une épreuve n'est plus assez de participants, on la supprime.

## Utilisation de GIT

Ce rapport et tous les codes sont sur github.