Groupe 4
De Viala Rosales, Gerard
Lombard, Antonin

CEBD: PROJET PARTIE 1

1. Introduction

Pour cette première partie du projet, on nous propose un modèle pour administrer les Jeux Olympiques. On nous propose deux relations concernant les sportifs participant aux JO et les épreuves qui se déroulent. Notre but est de compléter la conception proposé pour rendre une base de données fonctionnelle.

2. Conception UML et Normalisation

a. Question 1

Les deux relations sont indépendantes, il nous faudra donc créer une troisième relation pour lier les deux relations dans le futur ainsi que la création des tables supplémentaires pour éviter les doublons au maximum et rendre les relations BNCF. Afin de bien faire la conception, on définit les dépendances pour ensuite les normaliser (BNCF). Il n'existe pas une relation entre les deux tables.

Les dépendances qu'on peut déduire des relations sont les suivantes:

Pour les épreuves:
numEp -> nomEp, formeEp, categorieEp, nbSportifsEp, dateEp, nomDi
nomEp ->nomDi

Le numEp est un attribut primaire et la clé de la relation. L'attribut nomEp, n'est pas une clé de la relation, ce qui donne que la relation soit de forme 3NF. Pour résoudre ce problème nous allons créer une troisième relation contenant les noms des épreuves et les disciplines où elles appartiennent. Nous retrouvons les nouvelles relations:

R1(<u>numEp, nomEp</u>, formeEp, categorieEp, nbSportifsEp, dateEp, nomDi)
R21(<u>nomEp</u>, nomDi) avec nomEp comme clé de cette relation

Pour les sportifs:
numSp -> nomSp, prenomSp, pays, categorieSp, dateNaisSp, numEq
nomSp, prenomSp -> numSp, pays, categorieSp, dateNaisSp, numEq

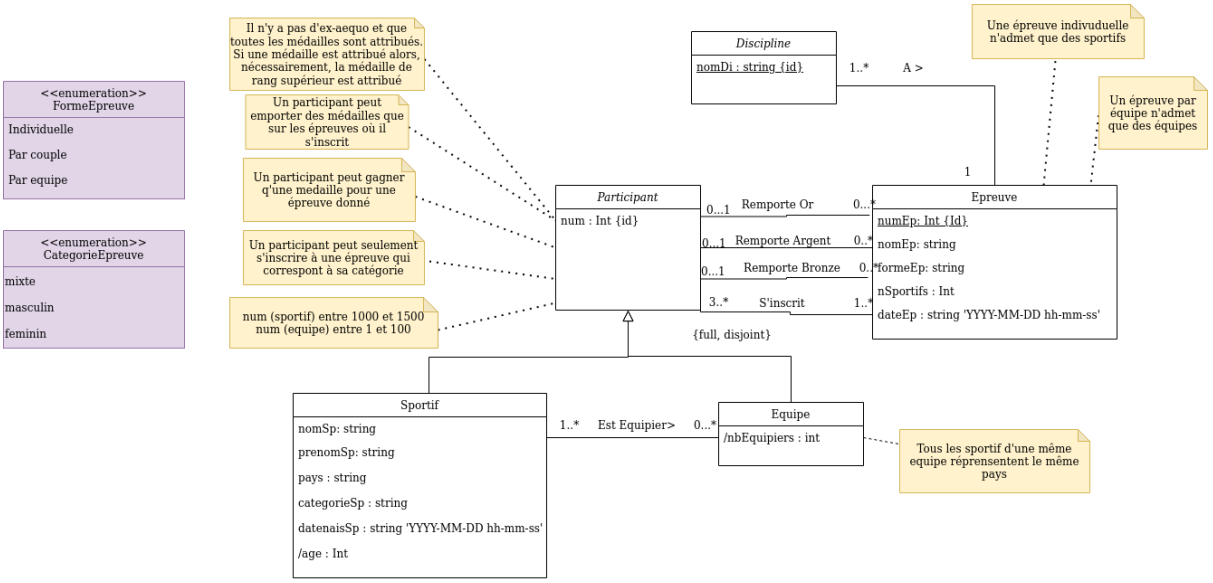
Le numSp, nomSp et prenomSp sont des attributs primaires. <u>numSp</u> et <u>nomSp, prenomSp</u> sont des clés de la relation sous forme BCNF.

b. Question 2

Dans cette partie on modélise le diagramme de classes UML des relations sous la forme BCNF. Après la normalisation nous avons pu définir des nouvelles relations avec plusieurs contraintes.

La catégorie d'un sportif et des épreuves vont liés, le sexe d'un participant définit dans quelles épreuves peut participer (categorieSp = {homme, femme}, categorieEp = {masculin, féminin, mixte}).

Si un sportif n'appartient pas à une équipe, il ne peut pas participer dans une épreuve par équipes.
Finalement tous les participants d'une même équipe font partie du même pays.



3. Implémentation

a. Question 3

Après la conversion du modèle UML vers un modèle relationnel on obtient les relations suivantes:

LesSportifs(<u>numSp</u>, nomSp, prenomSp, pays, categorieSp, datenaisSp)

/* <nsp, n, pren, p, c, d> ∈ LesSportifs ⇐⇒ Le sportif ayant le numéro nsp, de nom n et prénom pren, originaire du pays p et étant nais le d, appartenant à une catégorie c. Les sportifs ont une /age. */

LesEquipiers(<u>numSp, numEq</u>)

/* <nsp, neq> ∈ LesEquipiers ⇐⇒ Un sportif avec un numéro nsp, appartient à une équipe neq. */

LesEquipes(<u>numEq</u>)

/* <neq> ∈ LesEquipes ⇐⇒Les équipes sont identifiés par un numéro n. Les equipes ont un nombre d'équipiers /nbrEq. */

LesParticipants(<u>num</u>, numEp)

/* <n, nEp> ∈ LesParticipants ⇐⇒ Les participants identifiés par un numéro n sont inscrits à une épreuve numéro nEp */

LesDisciplines(nomDI)

/* <ndi> ∈ LesDisciplines ⇐⇒ Les disciplines sont identifiées par leur nom ndi. */

LesEpreuves(<u>numEp</u>, nomEp, formeEp, nbSportifs, dateEp,nomDi)

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

LesMedaillistesOr(numEp, num)

/* <n, nEp> ∈ LesMedaillistesOr ⇐⇒ Les gagnants d'une épreuve en première position sont identifiés par un numéro n et ont eu la première position à une épreuve de numéro nEp. */

LesMedaillistesArgent(numEp, num)

/* <n, nEp> ∈ LesMedaillistesArgent ⇐⇒ Les gagnants d'une épreuve en deuxième position sont identifiés par un numéro n et ont eu la deuxième position à une épreuve de numéro nEp */

LesMedaillistesBronze(numEp, num)

/* <n, nEp> ∈ LesMedaillistesBronze ⇐⇒ Les gagnants d'une épreuve en troisième position sont identifiés par un numéro n et ont eu la troisième position à une épreuve de numéro nEp. */

Les contraintes d'intégrité référentielles :

LesSportifs[numSp] ⊂ LesParticipants[num]
LesEquipes[numEq] ⊂ LesParticipants[num]

LesSportifs[numSp] ∪ LesEquipes[numEq] = LesParticipants[num]
LesSportifs[numSp] ∩ LesEquipes[numEq] = ∅

LesEquipiers[numSp] ⊆ LesSportifs[numSp]
LesEquipiers[numEq] = LesEquipes[numEq]

LesParticipants [numEp] ⊆ LesEpreuves[numEp]
Pour chaque épreuve nous devons avoir 3 participants (sportifs ou équipes) car toutes les médailles sont attribuées.

LesEpreuves[nomDi] = LesDisciplines[nomDi]

LesMedaillistes*[numEp] ⊆ LesEpreuves[numEp]
Si une médaille de catégorie supérieure est attribuée, alors la médaille de catégorie inférieure est aussi attribuée.

b. Question 4

i. Partie 1

- **Prise en main**

Pour comprendre le fonctionnement du programme fourni et le fonctionnement de chaque module (gui, data, utils et actions) nous devons remplacer la zone de saisie par une liste de valeurs prédéfinie.

Pour le faire nous devons modifier l'interface graphique en remplaçant la zone de saisie par une ComboBox avec les valeurs disponibles prédéfinis dans le fichier *fct_comp_1.ui*, ensuite respectant la convention de nommage proposée nous pouvons ensuite modifier le code python de la classe `class AppFctComp1Partie1(QDialog):` en remplaçant l'ancienne etiquette par `comboBox_fct_comp_1`.

Ensuite, en suivant la même procédure nous allons récupérer la liste de la comboBox d'après la base de données v0. Nous pouvons prendre la structure de la comboBox antérieure sans y définir les catégories. Pour récupérer les catégories nous modifions le constructeur de la classe `class AppFctComp2Partie1(QDialog):` en faisant une requête SQL pour les catégories `result = cursor.execute("SELECT DISTINCT categorieEp FROM V0_LesEpreuves")` qui nous permet de récupérer toutes les catégories distinctes.
On termine par remplacer les anciennes étiquettes par `comboBox_fct_comp_2` la nouvelle étiquette pour la deuxième comboBox.

- **Travail sur la BD**

Pour les **#TODO 1.3** nous allons travailler sur la base de données et on devra travailler sur tous les modules (interface graphique, le SQL et la récupération des données du fichier excel).

Pour le script SQL nous allons créer quatre tables extra (**LesEquipes, LesEquipiers, LesParticiants, LesResultats**) avec les **DROPS** correspondants. Nous rajoutons les vérifications **IF NOT EXISTS** dans pour les **CREATE TABLE** et les **DROP TABLE**.
La table **LesEquipes** contient le numéro des équipes.

La table **LesEquipiers** contient le numéro des équipes et les numéros des sportifs, avec les contraintes correspondantes. Dans cette table nous créons les deux **FOREIGN KEY numEq/numSp** et la clé primaire étant **(numEp, numSp)**.

La table **LesParticipants** c'est la table contenant le numéro des participants (**numP**) étant l'ensemble des numéros d'équipes et des sportifs. Pour faire cela la clé (**numP**) est définie comme une **FOREIGN KEY** qui fait **REFERENCES** les tables **LesEquipes(numEq)** ou **LesSportifs(numSp)**.

Finalement la table **LesResultats** contenant le numéro d'épreuve et le numéro de participant gagnant pour chaque médaille (Or, Argent, Bronze). D'après l'énoncé nous savons que chaque médaille doit être attribuée à la fin de chaque épreuve, on a donc ajouté ces contraintes vérifiant que les valeurs pour les médailles soit pas nul ou 0.

Les tables **LesSportif** et **LesEpreuves** ont été adaptées à notre schéma relationnel.

Nous avons modifié la fonction `def read_excel_file_V1(data:sqlite3.Connection, file):` en ajoutant la création pour chaque table existante dans notre base de base de données.

En suivant la procédure proposée dans l'extraction pour la base de données v0 nous avons rajouté les options pour ajouter les éléments correspondants pour chaque table. Vu que nous avons déjà créé les triggers pour éviter les doublons et les valeurs nulles dans la table équipiers.

- **Views**

Nous avons aussi créé les view pour calculer l'âge des sportifs et le nombre d'équipiers dans chaque équipe avec la requête SQL **CREATE VIEW**

Pour calculer l'âge des sportifs nous faisons une soustraction de la date actuelle et la date de naissance du Sportif (currentdate() - dateNaisSp).

```
-- TODO 1.4a : ajouter la définition de la vue LesAgesSportifs
CREATE VIEW IF NOT EXISTS LesAgesSportifs AS
    SELECT numSp, nomSp, prenomSp, pays, categorieSp, dateNaisSp, DATEDIFF(year, dateNaisSp, CURRENT_DATE)
    FROM LesSportifs;
```

Pour créer la table avec le nombre d'équipiers pour chaque équipe nous faisons un **COUNT(numSp)** des numéros de sportifs regroupés par le numéro d'équipe.

```
-- TODO 1.5a : ajouter la définition de la vue LesNbsEquipiers
CREATE VIEW IF NOT EXISTS LesNbsEquipiers AS
    SELECT numEq, COUNT(numSp)
    FROM LesEquipiers
    GROUP BY numEq;
```

ii. Partie 3

Nous avons rajouté les triggers permettant d'insérer et update les tables pour correspondre aux constrainte d'intégrité de notre schéma UML.