Rapport du projet de BDD3

Thomas Bignon, Hind Hamila
Binôme 4

Licence 2 Info, Groupe 1 & 2.

Schéma relationnel

La base de données est constituée de 12 tables. Les clés primaires sont soulignées et les clés secondaires sont suivi d'un #.

Athlete (IDAthlete, nomAthlete, Pays, IDSport#, Age)
Sport (IDSport, type, nomSport)
Equipe (IDEquipe, Pays, IDSport#, sexe)
ParticipationIndividuelle (IDParticipant, IDMatch#, statut, score)
ParticipationCollective (IDParticipant, IDMatch#, statut, score)
MatchCollectif (IDMatch, NomMatch, dateMatch, IDEpreuve#)
MatchIndividuel (IDMatch, NomMatch, dateMatch, IDEpreuve#)
Membres (IDAthlete#, IDEquipe#)
EpreuveCollective (IDEpreuve, IDSport, nomEpreuve, sexe, typeScore)
EpreuveIndividuelle (IDEpreuve, IDSport, nomEpreuve, sexe, typeScore)
MedailleCollective (IDMedaille, IDEpreuve, IDGagnant, type)
MedailleIndividuelle (IDMedaille, IDEpreuve, IDGagnant, type)

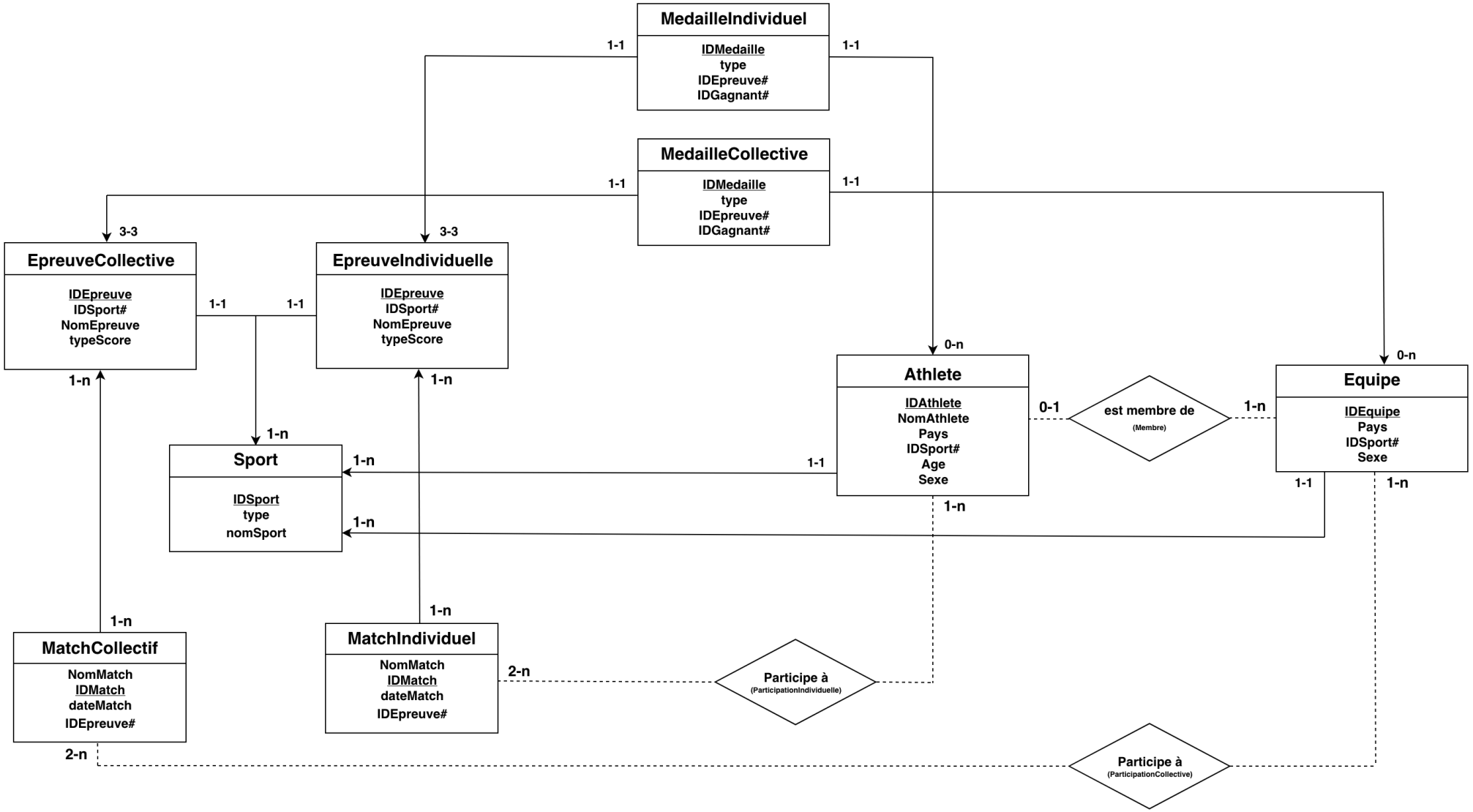
Fichiers

Le SQL a été séparé en 4 fichiers :

- '**arch.sql**' : Fichier qui construit les tables.
- '**data.sql**' : Fichier qui rempli les tables.
- '**requests.sql**' : Fichier qui contient les requêtes.
- '**prev.sql**' : Fichier qui modifie les tables et fais les requêtes de la partie 4.

Modèle Entités / Relations

(ci-dessous)


MedailleIndividuel
IDMedaille
type
IDEpreuve#
IDGagnant#
MedailleCollective
IDMedaille
type
IDEpreuve#
IDGagnant#
EpreuveCollective
IDEpreuve
IDSport#
NomEpreuve
typeScore
EpreuveIndividuelle
IDEpreuve
IDSport#
NomEpreuve
typeScore
Athlete
IDAthlete
NomAthlete
Pays
IDSport#
Age
Sexe
est membre de
(Membre)
Equipe
IDEquipe
Pays
IDSport#
Sexe
Sport
IDSport
type
nomSport
MatchCollectif
NomMatch
IDMatch
dateMatch
IDEpreuve#
MatchIndividuel
NomMatch
IDMatch
dateMatch
IDEpreuve#
Participe à
(ParticipationIndividuelle)
Participe à
(ParticipationCollective)
1-1
3-3
0-n
1-n
0-1
2-n

Choix de modélisation des tables

Nous avons d'abord opté pour un modèle simpliste composé de six tables différentes (Sport, Épreuve, Match, Athlète, Équipe et Médaille) toutefois cette configuration posait problème puisque la table épreuve devait à la fois intégrer des clés étrangères de la table Athlete ou de la table Equipe (IDAthlete et IDEquipe), nous avons donc scindé cette table en deux (EpreuveCollective et EpreuveIndividuelle).
La conséquence est que nous avons donc eu à faire la même chose pour les tables Match et Médaille plus tard pour cette même raison.

Après une première lecture des requêtes à faire, nous avons remarqué qu'un problème se posait concernant la liaison entre les tables Athlete et Equipe (nous ne savions pas quel Athlete était membre de quel équipe), nous avons donc créé une table Membre liant les ID des athlètes et des équipes auxquels ils appartiennent.
Le même problème s'est posé pour les matchs individuels et collectifs, nous avons donc créé deux tables 'ParticipationIndividuelle' et ' ParticipationCollective' qui lient les ID des participants aux matchs auxquels ils participent avec leur score en point ou en temps.

Nous avons pu pallier le problème concernant les scores de natures différentes en créant la colonne 'score' de type 'text'. Pour les requêtes demandant un temps ou un score, on a utilisé les fonctions 'CAST()' et SUBSTRING()' qui extrait et converti en int le score, le rendant utilisable pour des comparaisons d'entiers.

Nous avons fait le choix de ne pas mettre de champs 'not null' car cette base de données n'a pas été conçue pour être remplie par des utilisateurs mais seulement pour être consulté et pour plus de simplicité pour nous.

Nous avons fait le choix de ne pas utiliser de table Pays, puisqu'il est déjà intégré dans la colonne du même nom des tables Athlete et Equipe, réduisant le nombre total de tables.

Peuplement des tables

Bien que nous nous étions servi de fonctions java pour créer des équipes et des athlètes aléatoires, le peuplement des tables fut la tâche la plus longue à faire durant ce projet, plus de la moitié de notre temps de travail y a été consacrée.

Aussi, les tables Epreuve, Match, Médaille et Participation ont du être remplies à la main pour être cohérentes les unes avec les autres (par exemple l'athlète ayant gagné la médaille d'or en athlétisme devait avoir le meilleur score dans la table participation) et aussi pour écrire des noms d'épreuves et de matchs cohérents avec les sports liés. Plusieurs erreurs dans le remplissage de ces tables nous ont valu du temps de corrections supplémentaire.

Toutefois nous n'avons rempli les tables de Participation, d'épreuves et de matchs de façon partielle puisque la table Athlète disposait de plus de 700 tuples, nous avons donc remplis ces tables avec les épreuves nécessaires aux requêtes à faire ainsi que plusieurs autres tuples pour rendre les tables plus vivantes.

Difficultés rencontrées et limites du modèle utilisé

Mis à part le peuplement des tables, nous avons remarqué que notre modélisation du projet n'était pas adaptée à la requête concernant les médailles gagnées par Michael Phelps puisqu'il fallait aussi donner le temps correspondant à la médaille ce qui n'était pas possible avec notre conception. Il aurait fallu ajouter un champs IDParticipation qui serait un clé étrangère pour la table Participation qui contient le temps. Mais par manque de temps, à cause du fait qu'il aurait fallu un IDParticipation à chaque médaille, nous avons fait le choix de ne pas le rajouter.

À l'exception de cette issue, nous avons fait un usage facultatif des contraintes 'not null' et autre 'check constraints', facultatif car nous avions rempli les tables par nous même et que théoriquement une double correction n'était pas nécessaire.

Nous avons choisi d'utiliser la plateforme GitHub pour faire le projet. Cela a permis à mieux la maitriser (au prix de certaines pertes de bouts de code lors des 'merge') ce qui a permis en général de simplifier le partage du code entre nous et les modifications. Ainsi, le projet est disponible sur le repository 'totocptbgn/ProjetBDD'.