Projet CEBD

Syrine BEN HASSINE - Thibault GAILLARD - Antony FABRE
G1 - L3 info générale

Question 1

a) LesEpreuves

Relation :

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

Dépendances fonctionnelles trouvées :

- numEp → nomEp, formeEp, categorieEp, nbSportifsEp, dateEp, nomDi
- nomEp → nomDi
- (nomEp, formeEp) → nbSportifsEp

Clé : La clé est donc **numEp**.

Analyse des formes normales :

- La table est en **1NF** car chaque attribut contient une seule valeur.
- La table est en **2NF** car la clé génère tous les attributs non-clés.
- La table est en **3NF** car aucun attribut non-clé ne génère un autre attribut non-clé.
- La table n'est pas en **BCNF** car il existe les dépendances nomEp → nomDi et (nomEp, formeEp) → nbSportifsEp.

Décomposition en BCNF :

- T1(numEp {id}, nomEp, formeEp, categorieEp, nbSportifsEp, dateEp)
- T2(nomEp {id}, nomDi)
- T3(nomEp, formeEp {id double}, nbSportifsEp)

b) LesSportifsEQ

Relation :

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

Dépendances fonctionnelles trouvées :

- (nomSp, prenomSp, numEq) → numSp, pays, categorieSp, dateNaisSp
- numEq → pays
- numSp → nomSp, prenomSp, pays, categorieSp, dateNaisSp

Clés : Les clés de la table sont **(nomSp, numEq)**.

Analyse des formes normales :

- La table est en **1NF** car chaque attribut contient une seule valeur.
- La table n'est pas en **2NF** car (nomSp, prenomSp, categorieSp, dateNaisSp) ne dépend que de numSp.

Décomposition en BCNF :

- T1(nomSp, prenomSp, numEq {id triple}, numSp, pays, categorieSp, dateNaisSp)
- T2(numEq {id}, pays)
- T3(numSp {id}, nomSp, prenomSp, pays, categorieSp, dateNaisSp, numEq)

Diagramme UML et Contraintes

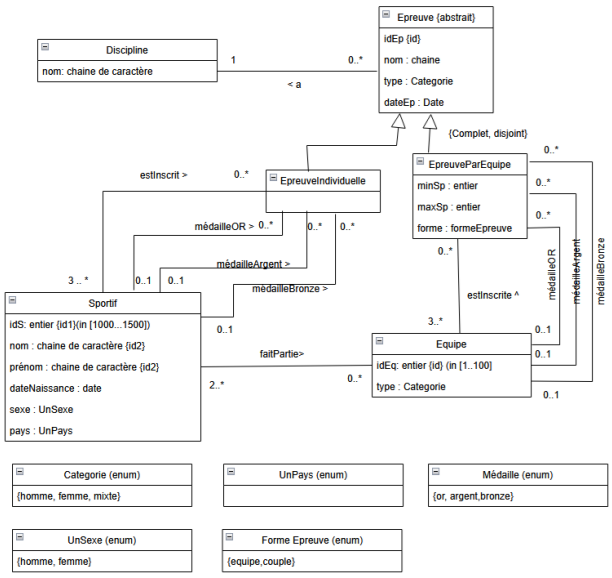

FIGURE 1 – Diagramme UML des entités principales

Contraintes supplémentaires (non représentées dans le diagramme UML)

- La date de naissance des sportifs est antérieure à la date de l'épreuve.
- Tous les sportifs d'une même équipe appartiennent au même pays.
- Pour une épreuve individuelle, la catégorie est du même type que le sexe du sportif.
- Pour une équipe féminine, il n'y a que des sportives féminines.
- Pour une équipe masculine, il n'y a que des sportifs masculins.
- Pour une équipe mixte, il y a au moins un sportif masculin et au moins une sportive féminine.
- Si une médaille est attribuée, alors nécessairement la médaille de rang supérieur est également attribuée.
- Les médailles (or, argent, bronze) d'une même épreuve sont attribuées à des personnes différentes.
- Une personne ou une équipe qui reçoit une médaille doit être inscrite à l'épreuve correspondante.

Question 3

On a décidé de faire de pays un type énuméré. Ce type décrit tous les pays et cela évite de faire des fautes de frappe en rentrant les données et ça évite à chaque JO de réécrire tous les pays.

On a aussi modifié Médaille. En suivant le schéma UML, on aurait dû avoir 3 tables : une pour les médailles d'or, une pour les médailles d'argent et une pour les médailles de bronze. Nous avons voulu rassembler en une grande table pour simplifier les requêtes et rassembler les données.
Une fois le schéma traduit, nous avons les relations suivantes (avec les contraintes associées).
LesPays(nomP) (type énuméré)

LesTypes(nomT) (type énuméré)

LesFormes(nomF) (type énuméré)

LesSexes(sexe) (type énuméré)

LesDisciplines(nomD)

LesEpreuvesIndividuelles(idEp, nom, type, date, discipline)
LesEpreuvesIndividuelles[discipline] $\subseteq$ LesDisciplines[nomD]
LesEpreuvesIndividuelles[type] $\subseteq$ LesTypes[nomT]

LesEpreuvesParEquipe(idEp, nom, type, date, minsp, maxsp, forme, discipline)
LesEpreuvesParEquipe[discipline] $\subseteq$ LesDisciplines[nomD]
LesEpreuvesParEquipe[forme] $\subseteq$ {'Equipe', 'Couple'}
LesEpreuvesParEquipe[type] $\subseteq$ {'Masculin', 'Feminin', 'Mixte'}
MinSP $\leq$ MaxSP

LesSportifs(idS, nom, prenom, date de naissance, sexe, pays)
nom, prénom clé secondaire
LesSportifs[pays] $\subseteq$ LesPays[nomP]
LesSportifs[sexe] $\subseteq$ {'Masculin', 'Feminin'}

LesEquipes(idEq, type)
LesEquipes[type] $\subseteq$ {'Masculin', 'Feminin', 'Mixte'}

LesMembresEquipes(idEq, idS)
LesMembresEquipes[idEq] $\subseteq$ LesEquipes[idEq]
LesMembresEquipes[idS] $\subseteq$ LesSportifs[idS]

LesInscriptionsEpreuvesIndividuelles(idEp, idS)
LesInscriptionsEpreuvesIndividuelles[idEp] $\subseteq$ LesEpreuvesIndividuelles[idEp]
LesInscriptionsEpreuvesIndividuelles[idS] $\subseteq$ LesSportifs[idS]

LesInscriptionsParEquipe(idEp, idEq)
LesInscriptionsParEquipe[idEp] $\subseteq$ LesEpreuvesParEquipe[idEp]
LesInscriptionsParEquipe[idEq] $\subseteq$ LesEquipes[idEq]

LesMedailles(idEp, or, argent, bronze)
LesMedailles[idEp] $\subseteq$ (LesInscriptionsEpreuvesIndividuelles[idEp] $\cup$ LesEpreuvesParEquipe[idEp])
LesMedailles[or] $\subseteq$ (LesMembresEquipes[idEq] $\cup$ LesInscriptionsEpreuvesIndividuelles[idEp])
LesMedailles[argent] $\subseteq$ (LesMembresEquipes[idEq] $\cup$ LesInscriptionsEpreuvesIndividuelles[idEp])
LesMedailles[bronze] $\subseteq$ (LesMembresEquipes[idEq] $\cup$ LesInscriptionsEpreuvesIndividuelles[idEp])

Vues possibles :
- Pour calculer le nombre de sportif dans une équipe
- Pour calculer l'âge des sportifs

Question 4

Étape 2 : Création des tables

```
DROP TABLE IF EXISTS LesInscriptionsEpreuvesParEquipes;
DROP TABLE IF EXISTS LesInscriptionsEpreuvesIndividuelles;
DROP TABLE IF EXISTS LesMedaillesEquipe;
DROP TABLE IF EXISTS LesMedaillesIndividuelle;
DROP TABLE IF EXISTS LesMembresEquipes;
DROP TABLE IF EXISTS LesEpreuvesParEquipe;
DROP TABLE IF EXISTS LesEpreuvesIndividuelles;
DROP TABLE IF EXISTS Discipline;
DROP TABLE IF EXISTS LesEquipes;
DROP TABLE IF EXISTS LesSportifs;

CREATE TABLE LesSportifs(
    idS         number(4),
    nom         VARCHAR2(20),
    prenom      VARCHAR2(20),
    date_nais   DATE,
    sexe        VARCHAR2(7),
    pays        VARCHAR2(20),
    CONSTRAINT SP_PK PRIMARY KEY (idS),
    CONSTRAINT SP_U1 UNIQUE (nom, prenom),
    CONSTRAINT SP_CK1 CHECK (sexe IN ('homme','femme')),
    CONSTRAINT SP_CK2 CHECK (idS > 999),
    CONSTRAINT SP_CK3 CHECK (idS < 1501)
);

CREATE TABLE LesEquipes(
    idEq    number(4),
    CONSTRAINT EQ_PK PRIMARY KEY (idEq),
    CONSTRAINT EQ_CK1 CHECK (idEq > 0),
    CONSTRAINT EQ_CK2 CHECK (idEq < 101)
);

CREATE TABLE Discipline(
    discipline VARCHAR2(25),
    CONSTRAINT DI_PK1 PRIMARY KEY(discipline)
);

CREATE TABLE LesEpreuvesIndividuelles(
    idEp        number(3),
    nom         VARCHAR2(25),
    type        VARCHAR2(7),
    dateEpi     DATE,
    discipline VARCHAR2(25),
    CONSTRAINT EPI_PK PRIMARY KEY (idEp),
    CONSTRAINT EPI_FK1 FOREIGN KEY (discipline) REFERENCES Discipline(discipline),
    CONSTRAINT EPI_CK CHECK (type IN ('homme', 'femme', 'mixte'))
);

CREATE TABLE LesEpreuvesParEquipe(
    idEp        NUMBER(4),
    nom         VARCHAR2(20),
    type        VARCHAR2(7),
    dateEpe     DATE,
    minsp       NUMBER(2),
    maxsp       NUMBER(2),
    forme       VARCHAR2(8),
    discipline VARCHAR2(25),
    CONSTRAINT EPE_PK1 PRIMARY KEY (idEp),
    CONSTRAINT EPE_FK1 FOREIGN KEY (discipline) REFERENCES Discipline(discipline),
    CONSTRAINT EPE_CK1 CHECK (type IN ('homme','femme','mixte')),
    CONSTRAINT EPE_CK2 CHECK (forme IN ('Equipe', 'Couple')),
    CONSTRAINT EPE_CK3 CHECK (minsp > 2),
    CONSTRAINT EPE_CK4 CHECK (minsp <= maxsp)
```

```
);

CREATE TABLE LesMembresEquipes(
    idEq NUMBER(4),
    idS  NUMBER(4),
    CONSTRAINT MEP_PK1 PRIMARY KEY (idEq, idS),
    CONSTRAINT MEP_FK1 FOREIGN KEY (idEq) REFERENCES LesEquipes(idEq),
    CONSTRAINT MEP_FK2 FOREIGN KEY (idS) REFERENCES LesSportifs(idS)
);

CREATE TABLE LesMedaillesIndividuelle(
    idEp   NUMBER(4),
    gold   NUMBER(4),
    argent NUMBER(4),
    bronze NUMBER(4),
    CONSTRAINT MI_PK1 PRIMARY KEY (idEp),
    CONSTRAINT MI_CK1 CHECK (gold != argent),
    CONSTRAINT MI_CK2 CHECK (bronze != argent),
    CONSTRAINT MI_CK3 CHECK (gold != bronze),
    CONSTRAINT MI_FK1 FOREIGN KEY (gold) REFERENCES LesSportifs(idS),
    CONSTRAINT MI_FK2 FOREIGN KEY (argent) REFERENCES LesSportifs(idS),
    CONSTRAINT MI_FK3 FOREIGN KEY (bronze) REFERENCES LesSportifs(idS),
    CONSTRAINT MI_CK4 CHECK (gold IS NOT NULL AND argent IS NOT NULL AND bronze IS NOT
    NULL)
);

CREATE TABLE LesMedaillesEquipe(
    idEp   NUMBER(4),
    gold   NUMBER(4),
    argent NUMBER(4),
    bronze NUMBER(4),
    CONSTRAINT EP_PK1 PRIMARY KEY (idEp),
    CONSTRAINT EP_CK1 CHECK (gold != argent AND argent != bronze AND gold != bronze),
    CONSTRAINT EP_FK1 FOREIGN KEY (gold) REFERENCES LesEquipes(idEq),
    CONSTRAINT EP_FK2 FOREIGN KEY (argent) REFERENCES LesEquipes(idEq),
    CONSTRAINT EP_FK3 FOREIGN KEY (bronze) REFERENCES LesEquipes(idEq),
    CONSTRAINT EP_CK2 CHECK (gold IS NOT NULL AND argent IS NOT NULL AND bronze IS NOT
    NULL)
);

CREATE TABLE LesInscriptionsEpreuvesIndividuelles(
    idEp NUMBER(4),
    idS  NUMBER(4),
    CONSTRAINT IEI_PK1 PRIMARY KEY (idEp, idS),
    CONSTRAINT IEI_FK1 FOREIGN KEY (idEp) REFERENCES LesEpreuvesIndividuelles(idEp),
    CONSTRAINT IEI_FK2 FOREIGN KEY (idS) REFERENCES LesSportifs(idS)
);

CREATE TABLE LesInscriptionsEpreuvesParEquipes(
    idEq NUMBER(4),
    idEp NUMBER(4),
    CONSTRAINT IPE_PK1 PRIMARY KEY (idEq, idEp),
    CONSTRAINT IPE_FK1 FOREIGN KEY (idEq) REFERENCES LesEquipes(idEq),
    CONSTRAINT IPE_FK2 FOREIGN KEY (idEp) REFERENCES LesEpreuvesParEquipe(idEp)
);
```

Étape 4 : Création des vues

```
DROP VIEW IF EXISTS LesAgesSportifs;
DROP VIEW IF EXISTS LesNbsEquipiers;
DROP VIEW IF EXISTS MoyenneAge;
DROP VIEW IF EXISTS ClassementPays;

CREATE VIEW LesAgesSportifs AS
SELECT
    idS,
    nom,
```

```
    prenom,
    pays,
    sexe,
    date_nais,
    CAST(
        (strftime('%Y', 'now') - strftime('%Y', date_nais))
        - (strftime('%m-%d', 'now') < strftime('%m-%d', date_nais))
    AS INTEGER) AS ageSp
FROM LesSportifs;

CREATE VIEW LesNbsEquipiers(numEq, nbEquipiersEq) AS
    SELECT idEq AS NumEq, COUNT(idS) AS nbEquipiersEq
    FROM LesMembresEquipes
    GROUP BY idEq;

CREATE VIEW MoyenneAge(idEq, moyAge) AS
    SELECT idEq, ROUND(AVG(ageSp),2) AS MoyAge
    FROM LesAgesSportifs
    JOIN LesMembresEquipes USING(idS)
    JOIN LesMedaillesEquipe ON (gold = idEq)
    GROUP BY idEq;

CREATE VIEW ClassementPays AS
SELECT
    s.pays,
    COUNT(mi_gold.idEp) + COUNT(me_gold.idEp) AS nbOr,
    COUNT(mi_argent.idEp) + COUNT(me_argent.idEp) AS nbArgent,
    COUNT(mi_bronze.idEp) + COUNT(me_bronze.idEp) AS nbBronze
FROM LesSportifs s
LEFT JOIN LesMedaillesIndividuelle mi_gold ON mi_gold.gold = s.idS
LEFT JOIN LesMedaillesEquipe me_gold ON me_gold.gold IN (SELECT idEq FROM
    LesMembresEquipes WHERE idS = s.idS)
LEFT JOIN LesMedaillesIndividuelle mi_argent ON mi_argent.argent = s.idS
LEFT JOIN LesMedaillesEquipe me_argent ON me_argent.argent IN (SELECT idEq FROM
    LesMembresEquipes WHERE idS = s.idS)
LEFT JOIN LesMedaillesIndividuelle mi_bronze ON mi_bronze.bronze = s.idS
LEFT JOIN LesMedaillesEquipe me_bronze ON me_bronze.bronze IN (SELECT idEq FROM
    LesMembresEquipes WHERE idS = s.idS)
GROUP BY s.pays
ORDER BY nbOr DESC, nbArgent DESC, nbBronze DESC;
```

Trigger : CheckInscriptionBeforeMedailleIndiv

```
-- ================================================================
-- TRIGGER: CheckInscriptionBeforeMedailleIndiv
-- OBJECTIF: Verifier les contraintes avant l'insertion d'une medaille individuelle
-- CONTRAINTES VeRIFIeES:
--   1. Le sportif doit etre inscrit    l'epreuve pour recevoir une medaille
--   2. Un sportif ne peut pas avoir deux medailles pour la meme epreuve
-- DeCLENCHEMENT: AVANT chaque INSERT dans LesMedaillesIndividuelles
-- ================================================================
CREATE OR REPLACE TRIGGER CheckInscriptionBeforeMedailleIndiv
BEFORE INSERT ON LesMedaillesIndividuelles  -- Table cible: medailles individuelles
FOR EACH ROW                                -- Execute pour chaque ligne inseree
DECLARE
    -- Variables pour stocker les resultats des verifications
    v_count_gold   NUMBER;   -- Nombre d'inscriptions trouvees pour le medaille d'or
    v_count_argent NUMBER;   -- Nombre d'inscriptions trouvees pour le medaille d'
    argent
    v_count_bronze NUMBER;   -- Nombre d'inscriptions trouvees pour le medaille de
    bronze
BEGIN
    -- ------------------------------------------------------------
    -- VeRIFICATION 1: Les sportifs doivent etre inscrits    l'epreuve
    -- ------------------------------------------------------------

```

```
    -- Verifier si le sportif medaille d'or est inscrit    cette epreuve
    -- COUNT(*) retourne 1 si inscrit, 0 sinon
    SELECT COUNT(*) INTO v_count_gold
    FROM LesInscriptionsEpreuvesIndividuelles  -- Table des inscriptions
    WHERE idEp = :NEW.idEp                     -- Meme epreuve que la medaille
      AND idS = :NEW.gold;                     -- Meme sportif que le medaille d'or

    -- Verifier si le sportif medaille d'argent est inscrit
    SELECT COUNT(*) INTO v_count_argent
    FROM LesInscriptionsEpreuvesIndividuelles
    WHERE idEp = :NEW.idEp
      AND idS = :NEW.argent;

    -- Verifier si le sportif medaille de bronze est inscrit
    SELECT COUNT(*) INTO v_count_bronze
    FROM LesInscriptionsEpreuvesIndividuelles
    WHERE idEp = :NEW.idEp
      AND idS = :NEW.bronze;

    -- Si un des trois sportifs n'est pas inscrit (count = 0), lever une exception
    -- Cette condition garantit que: Pour avoir une medaille, il faut etre inscrit
    IF v_count_gold = 0 OR v_count_argent = 0 OR v_count_bronze = 0 THEN
        RAISE_APPLICATION_ERROR(-20001,
            'Un ou plusieurs sportifs ne sont pas inscrits    cette epreuve
    individuelle');
    END IF;

    -- ============================================================
    -- VeRIFICATION 2: Pas de doublon de medaille pour un meme sportif
    -- ============================================================

    -- Verifier qu'un sportif n'a pas deux medailles pour la meme epreuve
    -- Cette condition compare les identifiants des sportifs entre eux
    -- Exemple: Si gold = argent, alors le meme sportif aurait or ET argent
    IF (:NEW.gold = :NEW.argent) OR
       (:NEW.gold = :NEW.bronze) OR
       (:NEW.argent = :NEW.bronze) THEN
        RAISE_APPLICATION_ERROR(-20002,
            'Un sportif ne peut pas avoir deux medailles pour la meme epreuve');
    END IF;

    -- Si on arrive ici, toutes les verifications sont passees
    -- L'insertion peut se poursuivre normalement
END;
/
```

Trigger : CheckInscriptionBeforeMedailleEquipe

```

-- ================================================================
-- TRIGGER: CheckInscriptionBeforeMedailleEquipe
-- OBJECTIF: Verifier les contraintes avant l'insertion d'une medaille par equipe
-- CONTRAINTES VeRIFIeES:
--   1. L'equipe doit etre inscrite    l'epreuve pour recevoir une medaille
--   2. Une equipe ne peut pas avoir deux medailles pour la meme epreuve
-- DeCLENCHEMENT: AVANT chaque INSERT dans LesMedaillesEquipe
-- ================================================================
CREATE OR REPLACE TRIGGER CheckInscriptionBeforeMedailleEquipe
BEFORE INSERT ON LesMedaillesEquipe          -- Table cible: medailles par equipe
FOR EACH ROW                                 -- Execute pour chaque ligne inseree
DECLARE
    -- Variables pour stocker les resultats des verifications
    v_count_gold   NUMBER;   -- Nombre d'inscriptions trouvees pour l'equipe d'or
    v_count_argent NUMBER;   -- Nombre d'inscriptions trouvees pour l'equipe d'argent
    v_count_bronze NUMBER;   -- Nombre d'inscriptions trouvees pour l'equipe de bronze
```

```
BEGIN
    -- ============================================================
    -- VeRIFICATION 1: Les equipes doivent etre inscrites    l'epreuve
    -- ============================================================

    -- Verifier si l'equipe medaillee d'or est inscrite    cette epreuve
    SELECT COUNT(*) INTO v_count_gold
    FROM LesInscriptionsEpreuvesParEquipes   -- Table des inscriptions par equipe
    WHERE idEp = :NEW.idEp                   -- Meme epreuve que la medaille
      AND idEq = :NEW.gold;                  -- Meme equipe que la medaille d'or

    -- Verifier si l'equipe medaillee d'argent est inscrite
    SELECT COUNT(*) INTO v_count_argent
    FROM LesInscriptionsEpreuvesParEquipes
    WHERE idEp = :NEW.idEp
      AND idEq = :NEW.argent;

    -- Verifier si l'equipe medaillee de bronze est inscrite
    SELECT COUNT(*) INTO v_count_bronze
    FROM LesInscriptionsEpreuvesParEquipes
    WHERE idEp = :NEW.idEp
      AND idEq = :NEW.bronze;

    -- Si une des trois equipes n'est pas inscrite (count = 0), lever une exception
    -- Cette condition garantit que: Pour avoir une medaille, il faut etre inscrit
    IF v_count_gold = 0 OR v_count_argent = 0 OR v_count_bronze = 0 THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Une ou plusieurs equipes ne sont pas inscrites    cette epreuve par equipe
    ');
    END IF;

    -- ============================================================
    -- VeRIFICATION 2: Pas de doublon de medaille pour une meme equipe
    -- ============================================================

    -- Verifier qu'une equipe n'a pas deux medailles pour la meme epreuve
    -- Cette condition compare les identifiants des equipes entre elles
    -- Exemple: Si gold = argent, alors la meme equipe aurait or ET argent
    IF (:NEW.gold = :NEW.argent) OR
       (:NEW.gold = :NEW.bronze) OR
       (:NEW.argent = :NEW.bronze) THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Une equipe ne peut pas avoir deux medailles pour la meme epreuve');
    END IF;

END;
/
```

Trigger : VerifierMemePaysEquipe

```

-- ======================================================================
-- TRIGGER: VerifierMemePaysEquipe
-- OBJECTIF: Verifier que tous les sportifs d'une meme equipe sont du meme pays
--           (contrainte explicitement mentionnee dans l'enonce)
-- eNONCe: "Tous les sportifs d'une meme equipe doivent etre du meme pays."
-- ======================================================================
CREATE OR REPLACE TRIGGER VerifierMemePaysEquipe
BEFORE INSERT OR UPDATE ON LesMembresEquipes
FOR EACH ROW
DECLARE
    v_pays_nouveau_membre VARCHAR2(20);
    v_pays_premier_membre VARCHAR2(20);
    v_nombre_membres NUMBER;
BEGIN
    -- 1. Recuperer le pays du nouveau sportif
```

```
    SELECT pays INTO v_pays_nouveau_membre
    FROM LesSportifs
    WHERE idS = :NEW.idS;

    -- 2. Verifier s'il y a dej  des membres dans l'equipe
    SELECT COUNT(*) INTO v_nombre_membres
    FROM LesMembresEquipes
    WHERE idEq = :NEW.idEq;

    -- 3. Si ce n'est pas le premier membre de l'equipe
    IF v_nombre_membres > 0 THEN
        -- Recuperer le pays du premier membre de l'equipe
        SELECT DISTINCT s.pays INTO v_pays_premier_membre
        FROM LesSportifs s
        JOIN LesMembresEquipes me ON s.idS = me.idS
        WHERE me.idEq = :NEW.idEq
        AND ROWNUM = 1;  -- Premier membre trouve

        -- 4. Verifier que le nouveau membre a le meme pays
        IF v_pays_nouveau_membre != v_pays_premier_membre THEN
            RAISE_APPLICATION_ERROR(-20015,
                'Le sportif ' || :NEW.idS || ' (pays: ' || v_pays_nouveau_membre ||
                ') ne peut pas rejoindre l''equipe ' || :NEW.idEq ||
                ' car les membres actuels sont du pays: ' || v_pays_premier_membre ||
                '. Tous les membres d''une equipe doivent etre du meme pays.');
        END IF;
    END IF;
END;
/
```

Explication Code Trigger

1. Trigger : `CheckInscriptionBeforeMedailleIndiv`

Objectif : Vérifier les contraintes avant l'insertion d'une médaille individuelle.
Ce trigger est exécuté **avant** chaque insertion dans la table `LesMedaillesIndividuelles`, et son rôle est de vérifier deux contraintes principales :

— **Vérification 1 : Les sportifs doivent être inscrits à l'épreuve** Avant d'attribuer une médaille à un sportif, il faut vérifier qu'il est inscrit à l'épreuve correspondante. Le trigger fait cela pour les trois médailles : or, argent et bronze. Il effectue trois requêtes pour chaque sportif (médaille d'or, argent et bronze) en comptant le nombre d'inscriptions de ce sportif à cette épreuve. Si un des sportifs n'est pas inscrit (`COUNT(*) = 0`), une erreur est levée.

— **Vérification 2 : Pas de doublon de médaille pour un même sportif** Un sportif ne peut pas recevoir deux médailles pour la même épreuve. Le trigger vérifie si un même sportif est attribué à plusieurs médailles (or, argent, bronze). Si c'est le cas, une exception est levée.

2. Trigger : `CheckInscriptionBeforeMedailleEquipe`

Objectif : Vérifier les contraintes avant l'insertion d'une médaille par équipe.
Ce trigger est similaire au précédent, mais il est destiné aux médailles par équipe, en vérifiant aussi deux contraintes principales :

— **Vérification 1 : Les équipes doivent être inscrites à l'épreuve** Avant d'attribuer une médaille à une équipe, il faut s'assurer que l'équipe est inscrite à l'épreuve. Le trigger effectue trois requêtes similaires pour vérifier l'inscription des trois équipes : or, argent et bronze.

— **Vérification 2 : Pas de doublon de médaille pour une même équipe** Il faut aussi s'assurer qu'une équipe ne peut pas recevoir deux médailles dans la même épreuve. Le trigger vérifie si une équipe est déjà inscrite sous une médaille différente et lève une erreur si nécessaire.

3. Trigger : `VerifierMemePaysEquipe`

Objectif : Vérifier que tous les sportifs d'une même équipe sont du même pays.
Ce trigger vérifie qu'un sportif ne peut pas être ajouté à une équipe si les autres membres de cette équipe viennent d'un pays différent. Cette contrainte s'applique lors de l'insertion ou de la mise à jour d'un membre d'une équipe dans la table `LesMembresEquipes`.

- **Vérification 1 : Récupérer le pays du nouveau membre** Le trigger récupère le pays du sportif à ajouter à l'équipe (via une requête dans la table `LesSportifs`).
- **Vérification 2 : Vérifier s'il y a déjà des membres dans l'équipe** Si l'équipe a déjà des membres (`v_nombre_membres > 0`), on récupère le pays du premier membre de l'équipe.
- **Vérification 3 : Comparer le pays** Si le pays du nouveau membre est différent du pays du premier membre de l'équipe, une erreur est levée.

Récapitulatif des Points Clés des Triggers

- `CheckInscriptionBeforeMedailleIndiv` : Vérifie que chaque sportif (or, argent, bronze) est inscrit à l'épreuve et qu'il n'y a pas de doublon de médaille pour un même sportif.
- `CheckInscriptionBeforeMedailleEquipe` : Vérifie que chaque équipe (or, argent, bronze) est inscrite à l'épreuve et qu'il n'y a pas de doublon de médaille pour une même équipe.
- `VerifierMemePaysEquipe` : Vérifie que tous les membres d'une équipe sont du même pays avant d'ajouter un nouveau membre à cette équipe.

Lien de GitHub

Le lien de depot git : https ://github.com/Syrinbh/ProjetCebd.git