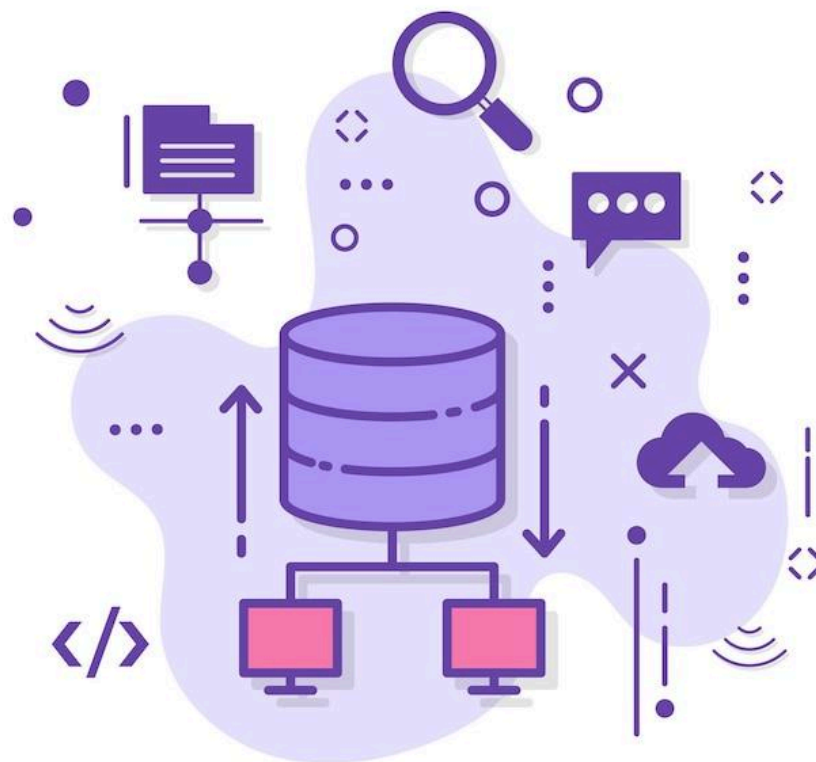


Baco Alistair 2B

Compte Rendu S204 Exploitation d'une BD : Partie Individuelle



université
PARIS-SACLAY

IUT D'ORSAY

2023-2024

1)Requêtes PL/SQL	2
i) Gestion des droits	2
a) Analyse	3
b) Gestion	3
ii) Vues imposées	4
iii) Fonctions imposées	8
a) JSON	8
b) Procédures d'Ajout	10
b.1) Individuel	10
b.2) Equipe	15
iv) Déclencheurs imposés	19
a) La Table LOG	19
b) Les Triggers	19
b.1)Triggers Imposés	19
b.1.1) Athlète	19
b.1.2)Composition_Equipe	21
b.1.3)Discipline	22
b.1.4)Hote	23
b.1.5)Evenement	25
b.1.6)Equipe	26
b.1.7)NOC	27
b.1.8)Participation_Equipe	28
b.1.9)Participation_Individuelle	29
b.1.10)Sport	31
b.2)Triggers Ajoutés	32
b.2.1)Session	32
b.2.2)Score_Individuel	34
b.2.3)Score_Equipe	35
2) Amélioration de la Base de Données	36
i)MCD & SR Améliorés	36
ii)Explication des Améliorations	37
iii) Exemples d'insertions et d'interrogations	38

1) Requêtes PL/SQL

Cette partie présente les différentes commandes PL/SQL demandées dans le sujet ainsi que les autres manières de manipulation de données que j'ai mis en place pour répondre aux consignes.

i) Gestion des droits

Le projet a deux acteurs principaux : AnalyseJO, qui peut consulter toute la base de données (sauf les Logs) et GestionJO, qui peut consulter et modifier la base de données ainsi que consulter les Logs (qui ne peuvent pas être modifiés pour des raisons évidentes).

La gestion des droits se sépare donc en deux parties de code :

a) Analyse

```
GRANT SELECT ON ATHLETE TO AnalyseJO;  
GRANT SELECT ON COMPOSITION_EQUIPE TO AnalyseJO;  
GRANT SELECT ON DISCIPLINE TO AnalyseJO;  
GRANT SELECT ON EQUIPE TO AnalyseJO;  
GRANT SELECT ON EVENEMENT TO AnalyseJO;  
GRANT SELECT ON HOTE TO AnalyseJO;  
GRANT SELECT ON NOC TO AnalyseJO;  
GRANT SELECT ON PARTICIPATION_INDIVIDUELLE TO AnalyseJO;  
GRANT SELECT ON PARTICIPATION_EQUIPE TO AnalyseJO;  
GRANT SELECT ON SPORT TO AnalyseJO;  
GRANT SELECT ON MEDAILLES_ATHLETES TO AnalyseJO;  
GRANT SELECT ON MEDAILLES_NOC TO AnalyseJO;  
REVOKE ALL ON LOG FROM AnalyseJO;  
GRANT EXECUTE ON biographie TO AnalyseJO;  
REVOKE ALL ON ajouter_resultat_individuel FROM AnalyseJO;  
REVOKE ALL ON ajouter_resultat_equipe FROM AnalyseJO;
```

b) Gestion

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ATHLETE TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON COMPOSITION_EQUIPE TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON DISCIPLINE TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON EQUIPE TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON EVENEMENT TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON HOTE TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON NOC TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON PARTICIPATION_INDIVIDUELLE TO  
GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON PARTICIPATION_EQUIPE TO  
GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON SPORT TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON PARTICIPATION_EQUIPE TO  
GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON MEDAILLES_NOC TO GestionJO;  
GRANT SELECT, INSERT, UPDATE, DELETE ON MEDAILLES_ATHLETES TO GestionJO;  
REVOKE ALL ON LOG FROM GestionJO;  
GRANT SELECT ON LOG TO GestionJO;GRANT EXECUTE ON  
ajouter_resultat_individuel TO GestionJO;  
GRANT EXECUTE ON ajouter_resultat_equipe TO GestionJO;  
GRANT EXECUTE ON biographie TO GestionJO;
```

ii) Vues imposées

Vue donnant la liste de tous les athlètes avec leurs médailles :

```
--Médailles pour les épreuves en solo, comptage individuel pour ensuite
additionner le tout, on sépare les deux cas pour limiter les duplicatas
liés aux join
CREATE OR REPLACE VIEW MEDAILLES_INDIVIDUELLES AS
SELECT
    IDATHLETE, SUM(CASE WHEN MEDAILLE = 'Gold' THEN 1 ELSE 0 END) AS
OR_MEDAILLES,
    SUM(CASE WHEN MEDAILLE = 'Silver' THEN 1 ELSE 0 END) AS
ARGENT_MEDAILLES,
    SUM(CASE WHEN MEDAILLE = 'Bronze' THEN 1 ELSE 0 END) AS
BRONZE_MEDAILLES,
    COUNT(MEDAILLE) AS TOTAL_MEDAILLES
FROM
    PARTICIPATION_INDIVIDUELLE pi
GROUP BY
    IDATHLETE;

--Médailles pour les épreuves en équipe
CREATE OR REPLACE VIEW MEDAILLES_EQUIPE AS
SELECT
    ce.IDATHLETE,
    SUM(CASE WHEN pe.MEDAILLE = 'Gold' THEN 1 ELSE 0 END) AS OR_MEDAILLES,
    SUM(CASE WHEN pe.MEDAILLE = 'Silver' THEN 1 ELSE 0 END) AS
ARGENT_MEDAILLES,
    SUM(CASE WHEN pe.MEDAILLE = 'Bronze' THEN 1 ELSE 0 END) AS
BRONZE_MEDAILLES,
    COUNT(pe.MEDAILLE) AS TOTAL_MEDAILLES
FROM
    PARTICIPATION_EQUIPE pe
INNER JOIN
    COMPOSITION_EQUIPE ce ON pe.IDEQUIPE = ce.IDEQUIPE
GROUP BY
    ce.IDATHLETE;

--Creation de la vue finale
CREATE OR REPLACE VIEW MEDAILLES_ATHLETES AS
SELECT
    a.IDATHLETE,
    a.NOMATHLETE,
```

```

a.PRENOMATHLETE,
-- Utilisation de NVL pour traiter les valeurs NULL comme 0
NVL(mi.OR_MEDAILLES, 0) + NVL(me.OR_MEDAILLES, 0) AS OR_MEDAILLES,
NVL(mi.ARGENT_MEDAILLES, 0) + NVL(me.ARGENT_MEDAILLES, 0) AS
ARGENT_MEDAILLES,
NVL(mi.BRONZE_MEDAILLES, 0) + NVL(me.BRONZE_MEDAILLES, 0) AS
BRONZE_MEDAILLES,
NVL(mi.TOTAL_MEDAILLES, 0) + NVL(me.TOTAL_MEDAILLES, 0) AS
TOTAL_MEDAILLES
FROM
    ATHLETE a
LEFT JOIN
    MEDAILLES_INDIVIDUELLES mi ON a.IDATHLETE = mi.IDATHLETE
LEFT JOIN
    MEDAILLES_EQUIPE me ON a.IDATHLETE = me.IDATHLETE
ORDER BY
    OR_MEDAILLES DESC,
    ARGENT_MEDAILLES DESC,
    BRONZE_MEDAILLES DESC,
    TOTAL_MEDAILLES DESC,
    a.NOMATHLETE ASC,
    a.PRENOMATHLETE ASC,
    a.IDATHLETE ASC;

```

Vue réalisant le même traitement pour les NOC :

```
--Medailles pour les épreuves en solo, comptage individuel pour ensuite
additionner le tout, on sépare les deux cas pour limiter les duplicatas
liés aux join
CREATE OR REPLACE VIEW MEDAILLES_INDIVIDUELLES_NOC AS
SELECT
    n.codenoc, SUM(CASE WHEN MEDAILLE = 'Gold' THEN 1 ELSE 0 END) AS
OR_MEDAILLES,
    SUM(CASE WHEN MEDAILLE = 'Silver' THEN 1 ELSE 0 END) AS
ARGENT_MEDAILLES,
    SUM(CASE WHEN MEDAILLE = 'Bronze' THEN 1 ELSE 0 END) AS
BRONZE_MEDAILLES,
    COUNT(MEDAILLE) AS TOTAL_MEDAILLES
FROM
    PARTICIPATION_INDIVIDUELLE pi
INNER JOIN
    noc n on pi.noc=n.codenoc
GROUP BY
    n.codenoc;

--Medailles pour les épreuves en équipe
CREATE OR REPLACE VIEW MEDAILLES_EQUIPE_NOC AS
SELECT
    n.codenoc,
    SUM(CASE WHEN pe.MEDAILLE = 'Gold' THEN 1 ELSE 0 END) AS OR_MEDAILLES,
    SUM(CASE WHEN pe.MEDAILLE = 'Silver' THEN 1 ELSE 0 END) AS
ARGENT_MEDAILLES,
    SUM(CASE WHEN pe.MEDAILLE = 'Bronze' THEN 1 ELSE 0 END) AS
BRONZE_MEDAILLES,
    COUNT(pe.MEDAILLE) AS TOTAL_MEDAILLES
FROM
    PARTICIPATION_EQUIPE pe
INNER JOIN
    equipe e on pe.idequipe=e.idequipe
inner join
    noc n on e.noc=n.codenoc
GROUP BY
    n.codenoc;

--Creation de la vue finale
CREATE OR REPLACE VIEW MEDAILLES_NOC AS
SELECT
```

```

        NVL(mi.OR_MEDAILLES, 0) + NVL(me.OR_MEDAILLES, 0) AS OR_MEDAILLES,
        NVL(mi.ARGENT_MEDAILLES, 0) + NVL(me.ARGENT_MEDAILLES, 0) AS
ARGENT_MEDAILLES,
        NVL(mi.BRONZE_MEDAILLES, 0) + NVL(me.BRONZE_MEDAILLES, 0) AS
BRONZE_MEDAILLES,
        NVL(mi.TOTAL_MEDAILLES, 0) + NVL(me.TOTAL_MEDAILLES, 0) AS
TOTAL_MEDAILLES
FROM
    noc n
LEFT JOIN
    MEDAILLES_INDIVIDUELLES_NOC mi ON n.codenoc = mi.codenoc
LEFT JOIN
    MEDAILLES_EQUIPE_NOC me ON n.codenoc = me.codenoc
ORDER BY
    OR_MEDAILLES DESC,
    ARGENT_MEDAILLES DESC,
    BRONZE_MEDAILLES DESC,
    TOTAL_MEDAILLES DESC,
    n.codenoc ASC;

```


iii) Fonctions imposées

a) JSON

J'ai décidé par manque de temps de me limiter à "biographie" dans les fonctions retournant des objets JSON.

```
CREATE OR REPLACE FUNCTION biographie (id_athlete IN NUMBER) RETURN CLOB
AS
    obj_json clob;
    gold int;
    argent int;
    bronze int;
    total int;
BEGIN
    -- Vérifier si l'athlète existe
    SELECT COUNT(*) INTO total FROM Athlete WHERE idathlete =
id_athlete;
    IF total = 0 THEN
        -- Lancer une exception si l'athlète n'existe pas
        RAISE_APPLICATION_ERROR(-20011, 'Athlète inconnu');
    END IF;

    -- Sélectionner les médailles de l'athlète
    SELECT OR_MEDAILLES, ARGENT_MEDAILLES, BRONZE_MEDAILLES,
TOTAL_MEDAILLES
    INTO gold, argent, bronze, total
    FROM MEDAILLES_ATHLETES
    WHERE MEDAILLES_ATHLETES.idathlete = id_athlete;

    -- Sélectionner les informations de l'athlète
    SELECT JSON_OBJECT(
        'nom'      VALUE NomAthlete,
        'prénom'   VALUE PrenomAthlete,
        'surnom'    VALUE Surnom,
        'genre'     VALUE SUBSTR(Genre, 1, 1),
        'dateNaissance' VALUE TO_CHAR(DateNaissance,
'YYYY-MM-DD'),
        'dateDécès'   VALUE TO_CHAR(DateDeces, 'YYYY-MM-DD'),
        'taille'      VALUE Taille || ' cm',
        'poids'       VALUE Poids || ' kg',
        'médaillesOr'  VALUE TO_CHAR(gold),
        'médaillesArgent' VALUE TO_CHAR(argent),
        'médaillesBronze' VALUE TO_CHAR(bronze),
```

```

        'médaillTotal'    VALUE TO_CHAR(total)
    )
    INTO obj_json
    FROM Athlete
    WHERE idathlete = id_athlete;

    RETURN obj_json;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Gérer l'exception si aucune ligne n'est trouvée
        RAISE_APPLICATION_ERROR(-20011, 'Athlète inconnu');
END;
/

```

b) Procédures d'Ajout

b.1) Individuel

```
CREATE OR REPLACE PROCEDURE ajouter_resultat_individuel(
    id_evenement NUMBER,
    id_athlete NUMBER,
    code_noc VARCHAR2,
    resultat VARCHAR2
)
AS
    varCheckPres NUMBER(10);
    varCheckEven BOOLEAN := FALSE;
    varCheckAthl BOOLEAN := FALSE;
    varCheckNOC BOOLEAN := FALSE;
    resultValid BOOLEAN := TRUE;
    resultegalvalid BOOLEAN := FALSE;
    eventStatus VARCHAR2(30);
    medailleObtenue VARCHAR2(10);
    NOCAthlete VARCHAR2(10);

    CURSOR medailles IS
    SELECT pi.resultat result, medaille
    FROM participation_individuelle pi
    WHERE idevent = id_evenement;

    CURSOR liste_resultat IS
    SELECT pi.resultat
    FROM participation_individuelle pi
    WHERE idevent = id_evenement;

    CURSOR participants_event IS
    SELECT idathlete
    FROM participation_individuelle pi
    WHERE idevent = id_evenement;

    CURSOR liste_athlete IS
    SELECT idathlete
    FROM athlete;

    CURSOR liste_event IS
    SELECT idevenement
    FROM evenement;

    CURSOR liste_noc IS
    SELECT codenoc
    FROM noc;
```

```

-- Déclaration de l'exception
INEXISTANT EXCEPTION;
PRAGMA EXCEPTION_INIT(INEXISTANT, -20001);
POSOCCUPEE EXCEPTION;
PRAGMA EXCEPTION_INIT(POSOCCUPEE, -20002);
DEJAPRESENT EXCEPTION;
PRAGMA EXCEPTION_INIT(DEJAPRESENT, -20003);
NOTNOC EXCEPTION;
PRAGMA EXCEPTION_INIT(NOTNOC, -20004);

BEGIN
    -- verification de l'existence de l'athlete, l'evenement et du noc

    -- Athlete
    FOR ligne IN liste_athlete
    LOOP
        IF id_athlete = ligne.idathlete THEN
            varCheckAthl := TRUE;
        END IF;
    END LOOP;

    IF varCheckAthl = FALSE THEN
        RAISE INEXISTANT;
    END IF;

    -- Evenement
    FOR ligne IN liste_event
    LOOP
        IF id_evenement = ligne.idevenement THEN
            varCheckEven := TRUE;
        END IF;
    END LOOP;

    IF NOT varCheckEven THEN
        RAISE INEXISTANT;
    END IF;

    -- NOC
    FOR ligne IN liste_noc
    LOOP
        IF code_noc = ligne.codenoc THEN
            varCheckNOC := TRUE;
        END IF;
    END LOOP;

```

```

    IF NOT varCheckNOC THEN
    RAISE INEXISTANT;
    END IF;

    -- verification de si l'athlete a déjà un résultat pour ladite
    épreuve
    FOR ligne IN participants_event
    LOOP
    IF id_athlete = ligne.idathlete THEN
        RAISE DEJAPRESENT;
    END IF;
    END LOOP;

    -- Verification de la validité du résultat
    resultValid := TRUE;
    FOR ligneC IN liste_resultat
    LOOP
    IF SUBSTR(ligneC.resultat, 1, 1) = '=' THEN
        IF SUBSTR(ligneC.resultat, 1, 3) = SUBSTR(resultat, 1,
3) THEN
            resultValid := TRUE;
            resultegalvalid := TRUE;
        ELSE
            IF resultegalvalid = FALSE THEN
                resultValid := FALSE;
            END IF;
        END IF;
    ELSE
        IF SUBSTR(ligneC.resultat, 1, 2) = SUBSTR(resultat, 1,
2) THEN
            resultValid := FALSE;
        END IF;
    END IF;
    END LOOP;

    IF resultValid = FALSE THEN
    RAISE POSOCCUPEE;
    END IF;

    -- Mise en place de la medaille
    -- 1) verification de si l'evenement est bien olympique ou
    intercalé
    -- 2) application du texte en fonction de la medaille
    SELECT DISTINCT statutevenement
    INTO eventStatus
    FROM evenement e

```

```

WHERE e.idevenement = id_evenement;

IF eventStatus = 'Olympic' OR eventStatus = 'Intercalated' THEN
IF resultatvalid THEN
    IF SUBSTR(resultat, 1, 2) = '=1' THEN
        medailleObtenue := 'Gold';
    ELSIF SUBSTR(resultat, 1, 2) = '=2' THEN
        medailleObtenue := 'Silver';
    ELSIF SUBSTR(resultat, 1, 2) = '=3' THEN
        medailleObtenue := 'Bronze';
    ELSE
        medailleObtenue := NULL;
    END IF;
ELSE
    IF SUBSTR(resultat, 1, 1) = '1' THEN
        medailleObtenue := 'Gold';
    ELSIF SUBSTR(resultat, 1, 1) = '2' THEN
        medailleObtenue := 'Silver';
    ELSIF SUBSTR(resultat, 1, 1) = '3' THEN
        medailleObtenue := 'Bronze';
    ELSE
        medailleObtenue := NULL;
    END IF;
END IF;
ELSE
    medailleObtenue := NULL;
END IF;

-- Verification de la cohérence du NOC
SELECT DISTINCT noc
INTO NOCAthlete
FROM participation_individuelle
WHERE idathlete = id_athlete;

IF NOCAthlete IS NOT NULL THEN
IF NOCAthlete != code_noc THEN
    RAISE NOTNOC;
END IF;
END IF;

-- Insertion Finale
INSERT INTO participation_individuelle VALUES (id_athlete,
id_evenement, resultat, medailleObtenue, code_noc);

EXCEPTION
    WHEN INEXISTANT THEN

```

```

        IF varCheckAthl = false THEN RAISE_APPLICATION_ERROR(-20001,
'Athlète inexistant'); END IF;
        IF varCheckEven = false THEN RAISE_APPLICATION_ERROR(-20001,
'Événement inexistant'); END IF;
        IF varCheckNoc = false THEN RAISE_APPLICATION_ERROR(-20001, 'NOC
inexistant'); END IF;
        WHEN DEJAPRESENT THEN
        RAISE_APPLICATION_ERROR(-20003, 'Athlète déjà classé');
        WHEN POSOCCUPEE THEN
        RAISE_APPLICATION_ERROR(-20002, 'Position déjà occupée');
        WHEN NOTNOC THEN
        RAISE_APPLICATION_ERROR(-20004, 'Incohérence de NOC');
END;
/

```

b.2) Equipe

```
CREATE OR REPLACE PROCEDURE ajouter_resultat_equipe(
    id_evenement NUMBER,
    id_equipe NUMBER,
    resultat VARCHAR2
)
AS
    varCheckEven BOOLEAN := FALSE;
    varCheckEquipe BOOLEAN := FALSE;
    resultValid BOOLEAN := TRUE;
    resultegalvalid BOOLEAN := FALSE;
    eventStatus VARCHAR2(30);
    medailleObtenue VARCHAR2(10);

    CURSOR medailles IS
        SELECT pe.resultat result, medaille
        FROM participation_equipe pe
        WHERE idevenement = id_evenement;

    CURSOR liste_resultat IS
        SELECT pe.resultat
        FROM participation_equipe pe
        WHERE idevenement = id_evenement;

    CURSOR participants_event IS
        SELECT idequipe
        FROM participation_equipe pe
        WHERE idevenement = id_evenement;

    CURSOR liste_equipe IS
        SELECT idequipe
        FROM equipe;

    CURSOR liste_event IS
        SELECT idevenement
        FROM evenement;

    -- Déclaration de l'exception
    INEXISTANT EXCEPTION;
    PRAGMA EXCEPTION_INIT(INEXISTANT, -20001);
    POSOCCUPEE EXCEPTION;
    PRAGMA EXCEPTION_INIT(POSOCCUPEE, -20002);
    DEJAPRESENT EXCEPTION;
    PRAGMA EXCEPTION_INIT(DEJAPRESENT, -20003);

BEGIN
```



```

-- verification de l'existence de l'équipe et de l'événement

-- Équipe
FOR ligne IN liste_equipe
LOOP
IF id_equipe = ligne.idequipe THEN
    varCheckEquipe := TRUE;
END IF;
END LOOP;

IF NOT varCheckEquipe THEN
RAISE INEXISTANT;
END IF;

-- Evenement
FOR ligne IN liste_event
LOOP
IF id_evenement = ligne.idevenement THEN
    varCheckEven := TRUE;
END IF;
END LOOP;

IF NOT varCheckEven THEN
RAISE INEXISTANT;
END IF;

-- verification de si l'équipe a déjà un résultat pour ledit
épreuve
FOR ligne IN participants_event
LOOP
IF id_equipe = ligne.idequipe THEN
    RAISE DEJAPRESENT;
END IF;
END LOOP;

-- Verification de la validité du résultat
resultValid := TRUE;
FOR ligneC IN liste_resultat
LOOP
IF SUBSTR(ligneC.resultat, 1, 1) = '=' THEN
    IF SUBSTR(ligneC.resultat, 1, 3) = SUBSTR(resultat, 1, 3)
THEN
        resultValid := TRUE;
        resultegalvalid := TRUE;
    ELSE
        IF resultegalvalid = FALSE THEN

```

```

        resultValid := FALSE;
        END IF;
    END IF;
ELSE
    IF SUBSTR(ligneC.resultat, 1, 2) = SUBSTR(resultat, 1, 2)
THEN
        resultValid := FALSE;
        END IF;
    END IF;
END LOOP;

IF resultValid = FALSE THEN
    RAISE POSOCCUPEE;
END IF;

-- Mise en place de la medaille
-- 1) verification de si l'evenement est bien olympique ou
intercalé
-- 2) application du texte en fonction de la medaille
SELECT DISTINCT statutevenement
INTO eventStatus
FROM evenement e
WHERE e.idevenement = id_evenement;

IF eventStatus = 'Olympic' OR eventStatus = 'Intercalated' THEN
    IF resultegalvalid THEN
        IF SUBSTR(resultat, 1, 2) = '=1' THEN
            medailleObtenue := 'Gold';
        ELSIF SUBSTR(resultat, 1, 2) = '=2' THEN
            medailleObtenue := 'Silver';
        ELSIF SUBSTR(resultat, 1, 2) = '=3' THEN
            medailleObtenue := 'Bronze';
        ELSE
            medailleObtenue := NULL;
        END IF;
    ELSE
        IF SUBSTR(resultat, 1, 1) = '1' THEN
            medailleObtenue := 'Gold';
        ELSIF SUBSTR(resultat, 1, 1) = '2' THEN
            medailleObtenue := 'Silver';
        ELSIF SUBSTR(resultat, 1, 1) = '3' THEN
            medailleObtenue := 'Bronze';
        ELSE
            medailleObtenue := NULL;
        END IF;
    END IF;
END IF;

```

```

ELSE
medailleObtenue := NULL;
END IF;

-- Insertion Finale
INSERT INTO participation_equipe VALUES (id_equipe, id_evenement,
resultat, medailleObtenue);

EXCEPTION
    WHEN INEXISTANT THEN
        IF varCheckEquipe = false THEN RAISE_APPLICATION_ERROR(-20001,
'Équipe inexistante'); END IF;
        IF varCheckEven = false THEN RAISE_APPLICATION_ERROR(-20001,
'Événement inexistant'); END IF;
    WHEN DEJAPRESENT THEN
        RAISE_APPLICATION_ERROR(-20003, 'Équipe déjà classée');
    WHEN POSOCCUPEE THEN
        RAISE_APPLICATION_ERROR(-20002, 'Position déjà occupée');
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20005, 'Erreur inconnue : ' || SQLERRM);
END;
/

```

iv) Déclencheurs imposés

a) La Table LOG

Pour mettre en place les déclencheurs, j'ai d'abord créé la table LOG avec les contraintes nécessaires avec une clef primaire s'incrémentent automatiquement pour ne pas avoir à le gérer dans l'activation de mes triggers, le code de sa création est le suivant :

```
CREATE TABLE LOG (  
    idLog NUMBER GENERATED ALWAYS AS IDENTITY,  
    idAuteur VARCHAR2(100) NOT NULL,  
    action VARCHAR2(20) NOT NULL,  
    dateHeureAction TIMESTAMP NOT NULL,  
    ligneAvant VARCHAR2(4000),  
    ligneApres VARCHAR2(4000),  
    PRIMARY KEY (idLog)  
);
```

b) Les Triggers

b.1) Triggers Imposés

Ci-dessous se trouvent les triggers imposés par le sujet

b.1.1) Athlète

```
CREATE OR REPLACE TRIGGER athleteTriggers  
AFTER INSERT OR DELETE OR UPDATE ON ATHLETE  
FOR EACH ROW  
DECLARE  
    auteurid NUMBER;  
    ancienneLigne VARCHAR2(4000);  
    nouvelleLigne VARCHAR2(4000);  
BEGIN  
    SELECT USER_ID  
    INTO auteurid  
    FROM ALL_USERS  
    WHERE USERNAME = USER;  
  
    IF INSERTING THEN
```

```

        ancienneLigne:=null;
        nouvelleLigne := :NEW.idAthlete ||',' ||:NEW.nomAthlete||
        ',' || :NEW.prenomAthlete || ',' || COALESCE(:NEW.SURNOM, '--') || ','
        || :NEW.GENRE ||','|| COALESCE(TO_CHAR(:NEW.DATENAISSANCE,
        'YYYY-MM-DD'), '--') || ',' || COALESCE(TO_CHAR(:NEW.DATEDECES,
        'YYYY-MM-DD'), '--') || ',' || COALESCE (:NEW.TAILLE,'--') || ',' ||
        COALESCE (:NEW.POIDS,'--');
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP,0,16),
        ancienneLigne, nouvelleLigne);
    END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.idAthlete ||',' ||:OLD.nomAthlete|| ',' ||
        :OLD.prenomAthlete || ',' || COALESCE(:OLD.SURNOM, '--') || ',' ||
        :OLD.GENRE ||','|| COALESCE(TO_CHAR(:OLD.DATENAISSANCE, 'YYYY-MM-DD'),
        '--') || ',' || COALESCE(TO_CHAR(:OLD.DATEDECES, 'YYYY-MM-DD'), '--')
        || ',' || COALESCE (:OLD.TAILLE,'--') || ',' || COALESCE
        (:OLD.POIDS,'--');
        nouvelleLigne := :NEW.idAthlete ||',' ||:NEW.nomAthlete || ',' ||
        :NEW.prenomAthlete|| ',' || COALESCE(:NEW.SURNOM, '--') || ',' ||
        :NEW.GENRE ||','|| COALESCE(TO_CHAR(:NEW.DATENAISSANCE, 'YYYY-MM-DD'),
        '--') || ',' || COALESCE(TO_CHAR(:NEW.DATEDECES, 'YYYY-MM-DD'), '--')||
        ',' || COALESCE (:NEW.TAILLE,'--') || ',' || COALESCE
        (:NEW.POIDS,'--');
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Mise a Jour', SUBSTR(CURRENT_TIMESTAMP,0,16),
        ancienneLigne, nouvelleLigne);
    END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.idAthlete ||',' ||:OLD.nomAthlete || ','
        ||:OLD.prenomAthlete || ',' || COALESCE(:OLD.SURNOM, '--') || ',' ||
        :OLD.GENRE ||','|| COALESCE(TO_CHAR(:OLD.DATENAISSANCE, 'YYYY-MM-DD'),
        '--') || ',' ||COALESCE(TO_CHAR(:OLD.DATEDECES, 'YYYY-MM-DD'), '--') ||
        ',' || COALESCE (:OLD.TAILLE,'--') || ',' || COALESCE (:OLD.POIDS,'--');
        nouvelleLigne := null;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP,0,16),
        ancienneLigne, nouvelleLigne);
    END IF;
END;
/

```

b.1.2)Composition_Equipe

```
CREATE OR REPLACE TRIGGER compoEquipeTrigger
AFTER INSERT OR DELETE OR UPDATE ON COMPOSITION_EQUIPE
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR2(4000);
    nouvelleLigne VARCHAR2(4000);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne:=null;
        nouvelleLigne := :NEW.idEquipe ||',' ||:NEW.idAthlete;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
        END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.idEquipe ||',' ||:OLD.idAthlete;
        nouvelleLigne := :NEW.idEquipe ||',' ||:NEW.idAthlete;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Mise a Jour', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
        END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.idEquipe ||',' ||:OLD.idAthlete;
        nouvelleLigne := null;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
        END IF;
END;
/
```

b.1.3)Discipline

```
CREATE OR REPLACE TRIGGER disciplineTrigger
AFTER INSERT OR DELETE OR UPDATE ON DISCIPLINE
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR2(4000);
    nouvelleLigne VARCHAR2(4000);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne:=null;
        nouvelleLigne := :NEW.codeDiscipline ||',' ||:NEW.NOMDISCIPLINE
        ||',' || :NEW.CODESPORT;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP,0,16),
        ancienneLigne, nouvelleLigne);
        END IF;

        IF UPDATING THEN
            ancienneLigne := :OLD.codeDiscipline ||',' ||:OLD.NOMDISCIPLINE
            ||',' || :OLD.CODESPORT;
            nouvelleLigne := :NEW.codeDiscipline ||',' ||:NEW.NOMDISCIPLINE
            ||',' || :NEW.CODESPORT;
            INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
            ligneApres) VALUES
            (auteurid, 'Mise a Jour', SUBSTR(CURRENT_TIMESTAMP,0,16),
            ancienneLigne, nouvelleLigne);
            END IF;

            IF DELETING THEN
                ancienneLigne := :OLD.codeDiscipline ||',' ||:OLD.NOMDISCIPLINE
                ||',' || :OLD.CODESPORT;
                nouvelleLigne := null;
                INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
                ligneApres) VALUES
                (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP,0,16),
                ancienneLigne, nouvelleLigne);
                END IF;
END;
/
```

b.1.4)Hote

```
CREATE OR REPLACE TRIGGER hoteTrigger
AFTER INSERT OR DELETE OR UPDATE ON HOTE
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR2(4000);
    nouvelleLigne VARCHAR2(4000);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne:=null;
        nouvelleLigne := :NEW.idhote ||',' ||:NEW.numerohote||',' ||
:NEW.LibelleHOTE||',' || :NEW.ANNEEHOTE||',' || :NEW.saison ||',' ||
:NEW.villehote ||',' || :new.codenochote
||',' ||COALESCE(:NEW.DATEOUVERTURE,
'--')||',' ||COALESCE(:NEW.DATEFERMETURE, '--') ||:NEW.DATESCOMPETITION
||',' || COALESCE(:NEW.NOTES, '--');
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
    END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.idhote ||',' ||:OLD.numerohote||',' ||
:OLD.LibelleHOTE||',' || :OLD.ANNEEHOTE||',' || :OLD.saison ||',' ||
:OLD.villehote ||',' || :OLD.codenochote
||',' ||COALESCE(:OLD.DATEOUVERTURE,
'--')||',' ||COALESCE(:OLD.DATEFERMETURE, '--') ||:OLD.DATESCOMPETITION
||',' || COALESCE(:OLD.NOTES, '--');
        nouvelleLigne := :NEW.idhote ||',' ||:NEW.numerohote||',' ||
:NEW.LibelleHOTE||',' || :NEW.ANNEEHOTE||',' || :NEW.saison ||',' ||
:NEW.villehote ||',' || :new.codenochote
||',' ||COALESCE(:NEW.DATEOUVERTURE,
'--')||',' ||COALESCE(:NEW.DATEFERMETURE, '--') ||:NEW.DATESCOMPETITION
||',' || COALESCE(:NEW.NOTES, '--');
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Mise a Jour', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
    END IF;
```



```

    IF DELETING THEN
        ancienneLigne := :OLD.idhote ||','|| :OLD.numerohote||','||
:OLD.LibelleHOTE||','|| :OLD.ANNEEHOTE||','|| :OLD.saison ||','||
:OLD.villehote ||','|| :OLD.codenochote
||','||COALESCE(:OLD.DATEOUVERTURE,
'--')||','||COALESCE(:OLD.DATEFERMETURE, '--') ||:OLD.DATESCOMPETITION
||','|| COALESCE(:OLD.NOTES, '--');
        nouvelleLigne := null;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
    END IF;
END;
/

```

b.1.5)Evenement

```
CREATE OR REPLACE TRIGGER evenementTrigger
AFTER INSERT OR DELETE OR UPDATE ON EVENEMENT
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR2(4000);
    nouvelleLigne VARCHAR2(4000);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne:=null;
        nouvelleLigne := :OLD.idevenement ||','
||:OLD.NOMEVENEMENT||','|| :OLD.STATUTEVENEMENT ||','||
:OLD.CODEDISCIPLINE || :OLD.IDHOTE;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
    END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.idevenement ||','
||:OLD.NOMEVENEMENT||','|| :OLD.STATUTEVENEMENT ||','||
:OLD.CODEDISCIPLINE || :OLD.IDHOTE;
        nouvelleLigne := :NEW.idevenement ||','
||:NEW.NOMEVENEMENT||','|| :NEW.STATUTEVENEMENT ||','||
:NEW.CODEDISCIPLINE || :NEW.IDHOTE;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Mise a Jour', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
    END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.idevenement ||',' ||:OLD.NOMEVENEMENT||','||
:OLD.STATUTEVENEMENT ||','|| :OLD.CODEDISCIPLINE || :OLD.IDHOTE;
        nouvelleLigne := null;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP,0,16),
ancienneLigne, nouvelleLigne);
    END IF;
```

```
END;  
/
```

b.1.6)Equipe

```
CREATE OR REPLACE TRIGGER equipeTrigger  
AFTER INSERT OR DELETE OR UPDATE ON EQUIPE  
FOR EACH ROW  
DECLARE  
    auteurid NUMBER;  
    ancienneLigne VARCHAR(100);  
    nouvelleLigne VARCHAR(100);  
BEGIN  
    SELECT USER_ID  
    INTO auteurid  
    FROM ALL_USERS  
    WHERE USERNAME = USER;  
  
    IF INSERTING THEN  
        ancienneLigne:=null;  
        nouvelleLigne := :NEW.idEQUIPE||',' ||:NEW.NOMEQUIPE||',' ||  
:NEW.NOC;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP,0,16),  
ancienneLigne, nouvelleLigne);  
    END IF;  
  
    IF UPDATING THEN  
        ancienneLigne := :OLD.idEQUIPE||',' ||:OLD.NOMEQUIPE||',' ||  
:OLD.NOC;  
        nouvelleLigne := :NEW.idEQUIPE||',' ||:NEW.NOMEQUIPE||',' ||  
:NEW.NOC;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Mise a Jour', SUBSTR(CURRENT_TIMESTAMP,0,16),  
ancienneLigne, nouvelleLigne);  
    END IF;  
  
    IF DELETING THEN  
        ancienneLigne := :OLD.idEQUIPE||',' ||:OLD.NOMEQUIPE||',' ||  
:OLD.NOC;  
        nouvelleLigne := null;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP,0,16),  
ancienneLigne, nouvelleLigne);  
    END IF;
```

```
END;  
/
```

b.1.7)NOC

```
CREATE OR REPLACE TRIGGER nocTrigger  
AFTER INSERT OR DELETE OR UPDATE ON noc  
FOR EACH ROW  
DECLARE  
    auteurid NUMBER;  
    ancienneLigne VARCHAR(100);  
    nouvelleLigne VARCHAR(100);  
BEGIN  
    SELECT USER_ID  
    INTO auteurid  
    FROM ALL_USERS  
    WHERE USERNAME = USER;  
  
    IF INSERTING THEN  
        ancienneLigne := NULL;  
        nouvelleLigne := :NEW.codenoc || ',' || :NEW.nomnoc;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneAprès) VALUES  
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
        END IF;  
  
    IF UPDATING THEN  
        ancienneLigne := :OLD.codenoc || ',' || :OLD.nomnoc;  
        nouvelleLigne := :NEW.codenoc || ',' || :NEW.nomnoc;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneAprès) VALUES  
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
        END IF;  
  
    IF DELETING THEN  
        ancienneLigne := :OLD.codenoc || ',' || :OLD.nomnoc;  
        nouvelleLigne := NULL;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneAprès) VALUES  
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
        END IF;  
END;  
/
```

b.1.8)Participation_Equipe

```
CREATE OR REPLACE TRIGGER participationEquipeTrigger
AFTER INSERT OR DELETE OR UPDATE ON participation_equipe
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR(200);
    nouvelleLigne VARCHAR(200);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne := NULL;
        nouvelleLigne := :NEW.idevenement || ',' || :NEW.idequipe || ','
        || :NEW.resultat || ',' || COALESCE(:NEW.medaille, 'NULL');
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
    END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.idevenement || ',' || :OLD.idequipe || ','
        || :OLD.resultat || ',' || COALESCE(:OLD.medaille, 'NULL');
        nouvelleLigne := :NEW.idevenement || ',' || :NEW.idequipe || ','
        || :NEW.resultat || ',' || COALESCE(:NEW.medaille, 'NULL');
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
    END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.idevenement || ',' || :OLD.idequipe || ','
        || :OLD.resultat || ',' || COALESCE(:OLD.medaille, 'NULL');
        nouvelleLigne := NULL;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
    END IF;
```

```
END;  
/
```

b.1.9)Participation_Individuelle

```
CREATE OR REPLACE TRIGGER participationIndividuelleTrigger  
AFTER INSERT OR DELETE OR UPDATE ON participation_individuelle  
FOR EACH ROW  
DECLARE  
    auteurid NUMBER;  
    ancienneLigne VARCHAR(200);  
    nouvelleLigne VARCHAR(200);  
BEGIN  
    SELECT USER_ID  
    INTO auteurid  
    FROM ALL_USERS  
    WHERE USERNAME = USER;  
  
    IF INSERTING THEN  
        ancienneLigne := NULL;  
        nouvelleLigne := :NEW.idathlete || ',' || :NEW.idevent || ',' ||  
:NEW.resultat || ',' || COALESCE(:NEW.medaille, 'NULL') || ',' ||  
:NEW.noc;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
    END IF;  
  
    IF UPDATING THEN  
        ancienneLigne := :OLD.idathlete || ',' || :OLD.idevent || ',' ||  
:OLD.resultat || ',' || COALESCE(:OLD.medaille, 'NULL') || ',' ||  
:OLD.noc;  
        nouvelleLigne := :NEW.idathlete || ',' || :NEW.idevent || ',' ||  
:NEW.resultat || ',' || COALESCE(:NEW.medaille, 'NULL') || ',' ||  
:NEW.noc;  
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
    END IF;  
  
    IF DELETING THEN  
        ancienneLigne := :OLD.idathlete || ',' || :OLD.idevent || ',' ||
```

```
:OLD.resultat || ',' || COALESCE(:OLD.medaille, 'NULL') || ',' ||  
:OLD.noc;  
    nouvelleLigne := NULL;  
    INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
    END IF;  
END;  
/
```

b.1.10)Sport

```
CREATE OR REPLACE TRIGGER sportTrigger
AFTER INSERT OR DELETE OR UPDATE ON sport
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR(100);
    nouvelleLigne VARCHAR(100);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne := NULL;
        nouvelleLigne := :NEW.codesport || ',' || :NEW.nomsport;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneAprès) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
ancienneLigne, nouvelleLigne);
        END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.codesport || ',' || :OLD.nomsport;
        nouvelleLigne := :NEW.codesport || ',' || :NEW.nomsport;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneAprès) VALUES
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
ancienneLigne, nouvelleLigne);
        END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.codesport || ',' || :OLD.nomsport;
        nouvelleLigne := NULL;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
ligneAprès) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
ancienneLigne, nouvelleLigne);
        END IF;
END;
/
```


b.2) Triggers Ajoutés

Ci-dessous se trouvent les triggers que j'ai rajouté suite à mon expansion de la base de données.

b.2.1) Session

```
CREATE OR REPLACE TRIGGER sessionEvenementTrigger
AFTER INSERT OR DELETE OR UPDATE ON SESSION_EVENEMENT
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR(200);
    nouvelleLigne VARCHAR(200);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne := NULL;
        nouvelleLigne := :NEW.TypeSession || ',' || :NEW.IdSession ||
        ',' || :NEW.IdEvent;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.TypeSession || ',' || :OLD.IdSession ||
        ',' || :OLD.IdEvent;
        nouvelleLigne := :NEW.TypeSession || ',' || :NEW.IdSession ||
        ',' || :NEW.IdEvent;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.TypeSession || ',' || :OLD.IdSession ||
```

```
' ,' || :OLD.IdEvent;  
    nouvelleLigne := NULL;  
    INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,  
ligneApres) VALUES  
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),  
ancienneLigne, nouvelleLigne);  
    END IF;  
END;  
/
```

b.2.2)Score_Individuel

```
CREATE OR REPLACE TRIGGER scoreIndividuelTrigger
AFTER INSERT OR DELETE OR UPDATE ON SCORE_INDIVIDUEL
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR(200);
    nouvelleLigne VARCHAR(200);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne := NULL;
        nouvelleLigne := :NEW.IdEvenement || ',' || :NEW.IdAthlete ||
        ',' || :NEW.IdSession || ',' || :NEW.resultatSession;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.IdEvenement || ',' || :OLD.IdAthlete ||
        ',' || :OLD.IdSession || ',' || :OLD.resultatSession;
        nouvelleLigne := :NEW.IdEvenement || ',' || :NEW.IdAthlete ||
        ',' || :NEW.IdSession || ',' || :NEW.resultatSession;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;

    IF DELETING THEN
        ancienneLigne := :OLD.IdEvenement || ',' || :OLD.IdAthlete ||
        ',' || :OLD.IdSession || ',' || :OLD.resultatSession;
        nouvelleLigne := NULL;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;
END;
/
```

b.2.3)Score_Equipe

```
CREATE OR REPLACE TRIGGER scoreEquipeTrigger
AFTER INSERT OR DELETE OR UPDATE ON SCORE_EQUIPE
FOR EACH ROW
DECLARE
    auteurid NUMBER;
    ancienneLigne VARCHAR(200);
    nouvelleLigne VARCHAR(200);
BEGIN
    SELECT USER_ID
    INTO auteurid
    FROM ALL_USERS
    WHERE USERNAME = USER;

    IF INSERTING THEN
        ancienneLigne := NULL;
        nouvelleLigne := :NEW.IdEvenement || ',' || :NEW.IdEquipe || ','
        || :NEW.IdSession || ',' || :NEW.resultatSession;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Insertion', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;

    IF UPDATING THEN
        ancienneLigne := :OLD.IdEvenement || ',' || :OLD.IdEquipe || ','
        || :OLD.IdSession || ',' || :OLD.resultatSession;
        nouvelleLigne := :NEW.IdEvenement || ',' || :NEW.IdEquipe || ','
        || :NEW.IdSession || ',' || :NEW.resultatSession;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Mise à Jour', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;

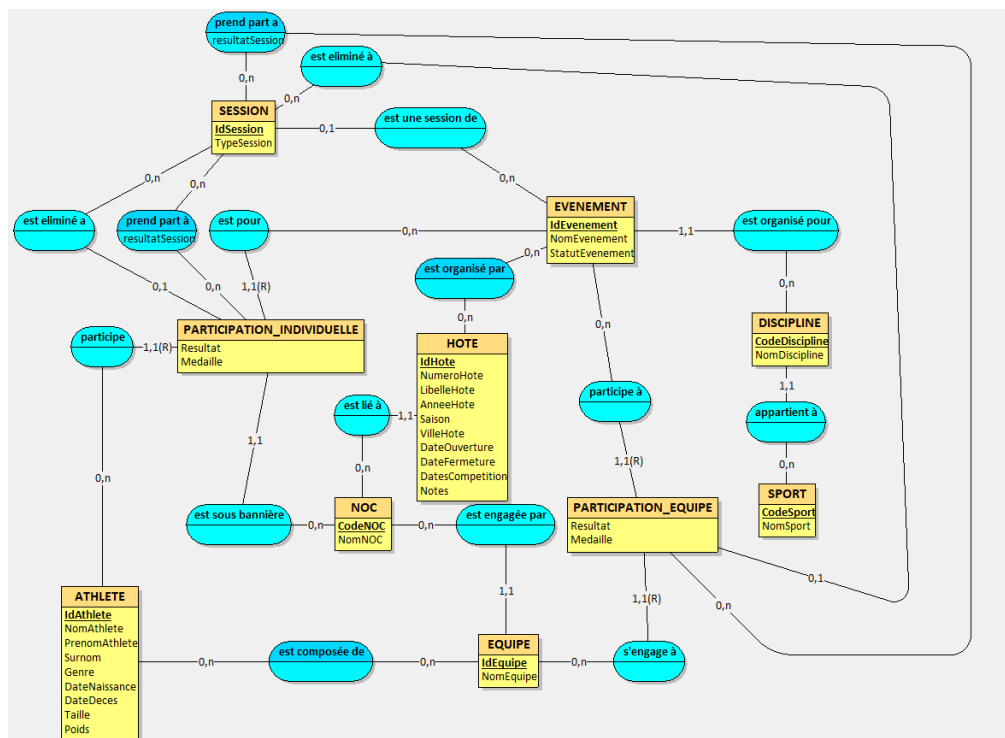
    IF DELETING THEN
        ancienneLigne := :OLD.IdEvenement || ',' || :OLD.IdEquipe || ','
        || :OLD.IdSession || ',' || :OLD.resultatSession;
        nouvelleLigne := NULL;
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant,
        ligneApres) VALUES
        (auteurid, 'Suppression', SUBSTR(CURRENT_TIMESTAMP, 0, 16),
        ancienneLigne, nouvelleLigne);
        END IF;
END;
/
```

2) Amélioration de la Base de Données

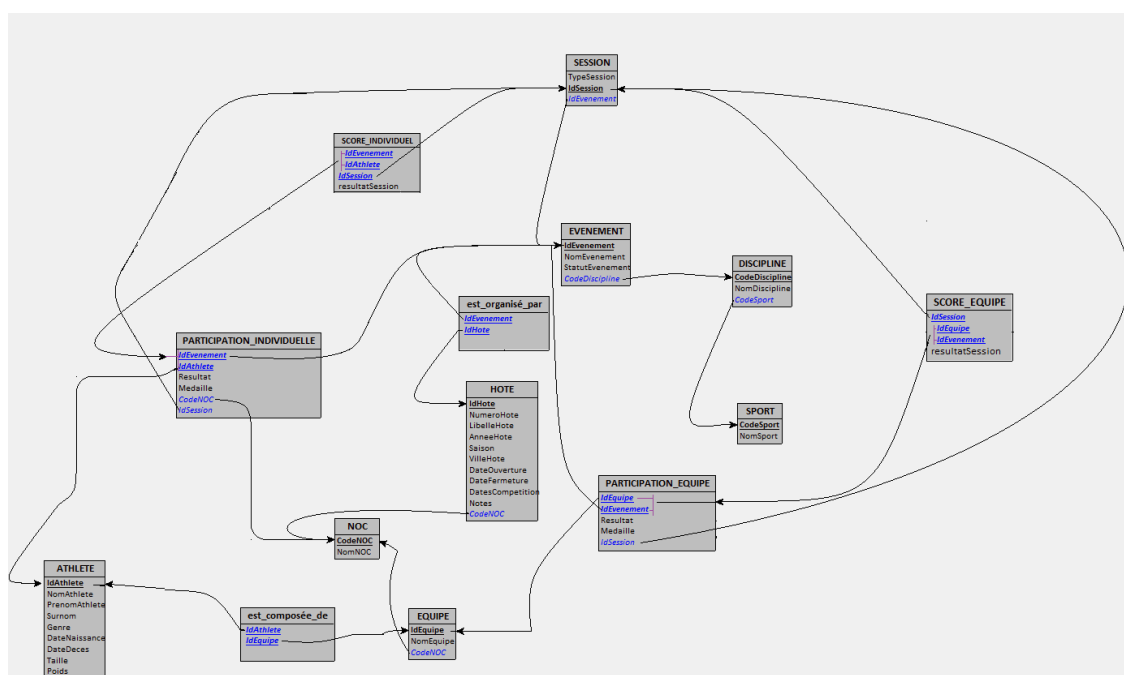
Dans cette seconde partie du sujet nous allons observer une manière d'implémenter la consultation détaillée des résultats.

i)MCD & SR Améliorés

MCD Augmenté :



SR Augmenté :



ii)Explication des Améliorations

Dans cette version augmentée de la base de données, une entité/table Session (qui a dû être renommée session_evenement dans le code SQL car SESSION est un mot réservé) est ajoutée et liée aux tables de participation des équipes et des athlètes individuels, mais aussi à un événement.

Ainsi, un événement possède plusieurs sessions qui ont elles-mêmes de multiples participants/équipes participantes. Les tables de participations ont désormais un attribut idSession, qui correspond à l'ID de la Session à laquelle l'athlète ou l'équipe a été éliminée dans l'épreuve. Cet attribut est donc NULL si l'athlète ou l'équipe a gagné l'épreuve.

Le résultat détaillé par session est disponible dans les tables SCORE_INDIVIDUEL et SCORE_EQUIPE grâce aux liaisons avec les tables de participations et la table des sessions. Chaque session est caractérisée non seulement par l'épreuve à laquelle elle appartient mais aussi par son type, attribut qui peut contenir des qualificatifs de la session tels que : « Finale, Demi-Finale, Quarts de Finale, Poules, Match amical, etc. »

iii) Exemples d'insertions et d'interrogations

Le script de création des tables est le suivant :









```
CREATE TABLE SESSION_EVENEMENT (  
    TypeSession VARCHAR(255),  
    IdSession INT PRIMARY KEY,  
    IdEvent INT,  
    FOREIGN KEY (IdEvent) REFERENCES evenement(IdEvenement)  
);  
  
CREATE TABLE SCORE_INDIVIDUEL (  
    IdEvenement INT,  
    IdAthlete INT,  
    IdSession INT,  
    resultatSession VARCHAR(255),  
    FOREIGN KEY (IdSession) REFERENCES SESSION_EVENEMENT(IdSession),  
    FOREIGN KEY (IdEvenement, IdAthlete) REFERENCES  
Participation_Individuelle(IdEvent, IdAthlete)  
);  
ALTER TABLE "SCORE_INDIVIDUEL" ADD CONSTRAINT "PK_SCORE_INDIVIDUEL"  
PRIMARY KEY ("IDSESSION", "IDATHLETE", "ID EVENEMENT") USING INDEX  
ENABLE;  
  
CREATE TABLE SCORE_EQUIPE (  
    IdSession INT,  
    IdEquipe INT,  
    IdEvenement INT,  
    resultatSession VARCHAR(255),  
    FOREIGN KEY (IdSession) REFERENCES SESSION_EVENEMENT(IdSession),  
    FOREIGN KEY (IdEvenement, IdEquipe) REFERENCES  
participation_equipe(IdEvenement, IdEquipe)  
);  
ALTER TABLE "SCORE_EQUIPE" ADD CONSTRAINT "PK_SCORE_EQUIPE" PRIMARY KEY  
("IDSESSION", "IDEQUIPE", "ID EVENEMENT") USING INDEX ENABLE;
```

Exemples d'Insertions de données :

Nous allons ici prendre l'exemple de la finale de barres asymétriques féminines :

Final (25 September 1988 — 12:40-13:20)

Top eight in individual competition advanced to apparatus final.

Pos	Competitor(s)	NOC	Points	QP(50%)	FP
1	Daniela Silivaş	 ROU	20.000	10.000	10.000
2	Dagmar Kersten	 GDR	19.987	9.987	10.000
3	Yelena Shushunova	 URS	19.962	9.962	10.000
4	Dörte Thümmel	 GDR	19.900	9.950	9.950
5	Sviatlana Bahinskaya	 URS	19.899	9.912	9.987
6	Iveta Poloková	 TCH	19.837	9.862	9.975
7	Aurelia Dobre	 ROU	19.824	9.862	9.962
8	Phoebe Mills	 USA	19.787	9.837	9.950

Pour incorporer les données de cette session uniquement dans la base nous allons suivre le procédé suivant (pour simplifier la chose prenons un idEvenement de 1) :

ici le Type de session est "Finale", l'id session est 2 puisque il n'y a eu quel les qualification avant la finale. La premiere ligne d'insertion est donc :

```
INSERT INTO SESSION VALUES ("Finale",2,1);
```

L'épreuve est individuelle nous allons donc ensuite enregistrer des données dans participation_individuelle (si elles n'étaient pas déjà renseignées)

```
--29034 est l'idAthlete de Daniela Silivas  
INSERT INTO PARTICIPATION_INDIVIDUELLE VALUES  
(1,29034,'1','Gold','ROU',NULL);
```

Enfin on peut renseigner le score dans SCORE_INDIVIDUEL

```
INSERT INTO SCORE_INDIVIDUEL VALUES (1,29034,2,'20.000');  
--A noter qu'il serait possible d'enregistrer la ligne de score complete  
dans "resultatSession" pour plus de précision, je décide de ne pas le  
faire pour simplifier la manipulation des données.
```

Une fois le processus répété pour chaque athlète et chaque session de l'évènement, l'épreuve est disponible avec ses scores détaillées dans la base de données.

Exemple d'exploitation de ces données :

Afficher le meilleur score obtenu aux barres asymétriques, pour potentiellement mettre à jour un classement des meilleurs athlètes de tous les temps

```
CREATE OR REPLACE VIEW record_barres_asymetrique AS
SELECT MAX(resultatsession) AS max_resultatsession
FROM score_individuel
WHERE idevenement = 1;
```

Voir les athlètes qui n'ont pas encore été éliminés d'une épreuve (part du principe que l'épreuve est toujours en cours, sinon cela montrerait les vainqueurs), cela permet de suivre en temps réel l'avancement des épreuves

```
CREATE OR REPLACE PROCEDURE athlete_en_lice (idevenement NUMBER) AS
    CURSOR enlice IS
        SELECT idathlete, nomathlete, prenomathlete
        FROM participation_individuelle
        NATURAL JOIN athlete
        WHERE idsession IS NULL AND idevenement =
athlete_en_lice.idevenement;
BEGIN
    FOR ligne IN enlice LOOP
        DBMS_OUTPUT.PUT_LINE('L''athlète ' || ligne.nomathlete || ' ' ||
ligne.prenomathlete || ' d''ID ' || ligne.idathlete || ' est encore en
lice.');
```