université PARIS-SACLAY

IUT D'ORSAY

Rapport S204

Partie 1

Taycir Ben Ouirane
Groupe 2 E

Fait le 2 juin 2024

Projet Base de données : Jeux olympiques

Table des matières

Taycir Ben Ouirane
Groupe 2E

1. Introduction

Dans le cadre du projet de base de données, j'ai importé les données à partir des fichiers .CSV de création de chaque table, en incluant les contraintes nécessaires. Ensuite, j'ai commencé par attribuer les droits aux utilisateurs GestionJO et AnalyseJO. J'ai étendu le schéma relationnel pour inclure les résultats détaillés des événements sportifs, en ajoutant les vues, fonctions et procédures recommandées par le professeur. De plus, j'ai mis en place plusieurs déclencheurs afin de faciliter la gestion des résultats pour les utilisateurs de la base de données. Par la suite, j'ai entamé la réflexion sur la partie 2 du projet, en prenant en compte les événements qui m'ont été attribués. Cela m'a conduit à étendre la base de données et à concevoir un nouveau schéma relationnel pour cette base étendue.

Partie Automatique : La partie automatique du projet a été évaluée régulièrement en interrogeant la vue NOTE_AUTO_S204 pour obtenir des retours sur les points d'évaluation attribués quotidiennement.

2. Requêtes et Blocs de Code

1. Gestion des droits

- **Donner les droits à AnalyseJO**

```
1. GRANT SELECT ON medailles_athletes.idathlete TO AnalyseJO;
2. GRANT SELECT ON SPORT TO AnalyseJO;
3. GRANT SELECT ON EVENEMENT TO AnalyseJO;
4. GRANT SELECT ON PARTICIPATION_EQUIPE TO AnalyseJO;
5. GRANT SELECT ON EQUIPE TO AnalyseJO;
6. GRANT SELECT ON PARTICIPATION_INDIVIDUELLE TO AnalyseJO;
7. GRANT SELECT ON COMPOSITION_EQUIPE TO AnalyseJO;
8. GRANT SELECT ON ATHLETE TO AnalyseJO;
9. GRANT SELECT ON NOC TO AnalyseJO;
10. GRANT SELECT ON HOTE TO AnalyseJO;
```

✓ **Vérification des droits de AnalyseJO**

```
1. SELECT * FROM USER_TAB_PRIVS WHERE GRANTEE = 'ANALYSEJO';
```

- **Donner les droits de modification à GestionJO**

```
1. GRANT SELECT, INSERT, UPDATE, DELETE ON DISCIPLINE TO GestionJO;
2. GRANT SELECT, INSERT, UPDATE, DELETE ON SPORT TO GestionJO;
3. GRANT SELECT, INSERT, UPDATE, DELETE ON EVENEMENT TO GestionJO;
4. GRANT SELECT, INSERT, UPDATE, DELETE ON PARTICIPATION_EQUIPE TO GestionJO;
5. GRANT SELECT, INSERT, UPDATE, DELETE ON EQUIPE TO GestionJO;
6. GRANT SELECT, INSERT, UPDATE, DELETE ON PARTICIPATION_INDIVIDUELLE TO GestionJO;
7. GRANT SELECT, INSERT, UPDATE, DELETE ON COMPOSITION_EQUIPE TO GestionJO;
8. GRANT SELECT, INSERT, UPDATE, DELETE ON ATHLETE TO GestionJO;
9. GRANT SELECT, INSERT, UPDATE, DELETE ON NOC TO GestionJO;
10. GRANT SELECT, INSERT, UPDATE, DELETE ON HOTE TO GestionJO;
```

✓ **Vérification des droits de GestionJO**

```
1. SELECT * FROM DBA_TAB_PRIVS WHERE GRANTEE = 'GESTIONJO';
```

2. Vues Imposées :

❖ **Vue Medailles_athletes**

```
2. CREATE OR REPLACE VIEW MEDAILLES_ATHLETES AS
3. SELECT
```


Taycir Ben Ouirane
Groupe 2E

```
    a.IdAthlete,
    a.NomAthlete,
    a.PrenomAthlete,
    SUM(CASE WHEN p.Medaille = 'Gold' THEN 1 ELSE 0 END) AS Nombre_Or,
    SUM(CASE WHEN p.Medaille = 'Silver' THEN 1 ELSE 0 END) AS Nombre_Argent,
    SUM(CASE WHEN p.Medaille = 'Bronze' THEN 1 ELSE 0 END) AS Nombre_Bronze,
    COUNT(p.Medaille) AS Nombre_Total
FROM
    ATHLETE a
LEFT JOIN (
    SELECT IdAthlete, Medaille FROM PARTICIPATION_INDIVIDUELLE
    UNION ALL
    SELECT ce.IdAthlete, pe.Medaille
    FROM PARTICIPATION_EQUIPE pe
    JOIN COMPOSITION_EQUIPE ce ON pe.IdEquipe = ce.IdEquipe
) p ON a.IdAthlete = p.IdAthlete
GROUP BY
    a.IdAthlete, a.NomAthlete, a.PrenomAthlete
ORDER BY
    Nombre_Or DESC,
    Nombre_Argent DESC,
    Nombre_Bronze DESC,
    Nombre_Total DESC,
    a.NomAthlete ASC,
    a.PrenomAthlete ASC,
    a.IdAthlete ASC;
```

✓ Droits utilisateurs

```
GRANT SELECT ON medailles_athletes TO AnalyseJO;
GRANT SELECT, INSERT, UPDATE, DELETE ON medailles_athletes TO GestionJO;
```

✓ Test

```
SELECT * FROM MEDAILLES_ATHLETES;
```

❖ Vue n°2 Medailles_Noc

```
CREATE OR REPLACE VIEW MEDAILLES_NOC AS
SELECT
    n.CodeNoc,
    SUM(CASE WHEN p.Medaille = 'Gold' THEN 1 ELSE 0 END) AS Nombre_Or,
    SUM(CASE WHEN p.Medaille = 'Silver' THEN 1 ELSE 0 END) AS Nombre_Argent,
    SUM(CASE WHEN p.Medaille = 'Bronze' THEN 1 ELSE 0 END) AS Nombre_Bronze,
    COUNT(p.Medaille) AS Nombre_Total
FROM
    noc n
LEFT JOIN (
    SELECT noc, Medaille FROM PARTICIPATION_INDIVIDUELLE
    UNION ALL
    SELECT e.noc, pe.Medaille
    FROM PARTICIPATION_EQUIPE pe
    JOIN EQUIPE e ON pe.IdEquipe = e.IdEquipe) p ON n.codenoc = p.noc
GROUP BY n.codeNoc, n.nomNoc
ORDER BY
    Nombre_Or DESC,
    Nombre_Argent DESC,
    Nombre_Bronze DESC,
    Nombre_Total DESC,
    n.codeNoc ASC;
```

✓ Droits utilisateurs

```
GRANT SELECT ON medailles_noc TO AnalyseJO;
GRANT SELECT, INSERT, UPDATE, DELETE ON medailles_noc TO GestionJO;
```

✓ Test

```
SELECT * FROM MEDAILLES_NOC;
```


Taycir Ben Ouirane
Groupe 2E

3. Fonctions Imposées :

❖ Fonction Biographie :

```
1.  CREATE OR REPLACE FUNCTION biographie(id_athlete INTEGER)
2.  RETURN VARCHAR2 AS
3.  athlete_json VARCHAR2(4000);
4.  v_nom ATHLETE.NOMATHLETE%TYPE;
5.  v_prenom ATHLETE.PRENOMATHLETE%TYPE;
6.  v_surnom ATHLETE.SURNOM%TYPE;
7.  v_genre ATHLETE.GENRE%TYPE;
8.  v_date_naissance ATHLETE.DATENAISSANCE%TYPE;
9.  v_date_deces ATHLETE.DATEDECES%TYPE;
10. v_taille ATHLETE.TAILLE%TYPE;
11. v_poids ATHLETE.POIDS%TYPE;
12. v_medailles_or INTEGER;
13. v_medailles_argent INTEGER;
14. v_medailles_bronze INTEGER;
15. v_medailles_total INTEGER;
16. BEGIN
17. athlete_json := '';
18. //Curseur pour récupérer les informations sur l'athlète
19. FOR ath_rec IN (SELECT NomAthlete, PrenomAthlete, Surnom, Genre, DateNaissance,
    DateDeces, Taille, Poids
20. FROM ATHLETE WHERE IdAthlete = id_athlete) LOOP
21. v_nom := ath_rec.NomAthlete;
22. v_prenom := ath_rec.PrenomAthlete;
23. v_surnom := ath_rec.Surnom;
24. v_genre := ath_rec.Genre;
25. v_date_naissance := ath_rec.DateNaissance;
26. v_date_deces := ath_rec.DateDeces;
27. v_taille := ath_rec.Taille;
28. v_poids := ath_rec.Poids;
29. END LOOP;
30. IF v_nom IS NULL THEN
31. //L'athlète n'existe pas
32. RAISE_APPLICATION_ERROR(-20011, 'Athlète inconnu');
33. END IF;
34. //Curseur pour récupérer les médailles de l'athlète
35. FOR medals_rec IN (SELECT COALESCE(Nombre_Or, 0) AS Nombre_Or,
36. COALESCE(Nombre_Argent, 0) AS Nombre_Argent,
37. COALESCE(Nombre_Bronze, 0) AS Nombre_Bronze,
38. COALESCE(Nombre_Total, 0) AS Nombre_Total
39. FROM MEDAILLES_ATHLETES
40. WHERE IdAthlete = id_athlete) LOOP
41. v_medailles_or := medals_rec.Nombre_Or;
42. v_medailles_argent := medals_rec.Nombre_Argent;
43. v_medailles_bronze := medals_rec.Nombre_Bronze;
44. v_medailles_total := medals_rec.Nombre_Total;
45. END LOOP;
46. //Construction de la chaîne JSON
47. athlete_json := '{' || '"nom": "' || v_nom || '",'
48. || '"prénom": "' || v_prenom || '",'
49. || '"surnom": "' || v_surnom || '",'
50. || '"genre": "' || v_genre || '",'
51. ||'"dateNaissance": ' || CASE  WHEN v_date_naissance IS NOT NULL THEN '"'
52. || TO_CHAR(v_date_naissance, 'YYYY-MM-DD') || '"'ELSE 'null' END
53. || ',' ||'"dateDécès": ' || CASE WHEN v_date_deces IS NOT NULL THEN '"'
54. || TO_CHAR(v_date_deces, 'YYYY-MM-DD') || '"'
55. ELSE 'null'END || ',' ||'"taille": "' || v_taille || ' cm",'
56. || '"poids": "' || v_poids || ' kg",'
57. || '"médaillesOr": ' || v_medailles_or || ','
58. || '"médaillesArgent": ' || v_medailles_argent || ','
59. || '"médaillesBronze": ' || v_medailles_bronze || ','
60. || '"médaillesTotal": ' || v_medailles_total || '}';
61. RETURN athlete_json;
```


Taycir Ben Ouirane
Groupe 2E

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
L'athlète n'existe pas
RAISE_APPLICATION_ERROR(-20011, 'Athlète inconnu');
END;
/
```

✓ **Droits utilisateurs**

```
GRANT execute ON biographie TO AnalyseJO;
GRANT execute ON biographie TO GestionJO;
```

✓ **Test**

```
SELECT biographie(124) FROM DUAL;
SELECT biographie(93860) FROM DUAL;
```

❖ Fonction Resultats

```
CREATE OR REPLACE FUNCTION resultats(id_evenement IN INTEGER)
RETURN t_resultats AS
v_resultats t_resultats := t_resultats();
BEGIN
FOR rec IN (
SELECT
JSON_OBJECT(
'position'   VALUE result.position,
'athlete(s)' VALUE result.athletes,
'noc'        VALUE result.noc,
'medaille'   VALUE result.medaille
) AS resultat
FROM (
SELECT
RANK() OVER (ORDER BY COALESCE(PI.Resultat, PE.Resultat), A.NomAthlete,
A.PrenomAthlete) AS position,
CASE
WHEN PE.IdEquipe IS NOT NULL THEN get_membres_equipe(PE.IdEquipe)
ELSE CAST(A.PrenomAthlete || ' ' || A.NomAthlete AS VARCHAR2(4000))
END AS athletes,
COALESCE(PI.NOC, EQ.NOC) AS noc,
COALESCE(PI.Medaille, PE.Medaille) AS medaille
FROM
EVENEMENT E
LEFT JOIN
PARTICIPATION_INDIVIDUELLE PI ON E.IdEvenement = PI.IdEvent
LEFT JOIN
ATHLETE A ON PI.IdAthlete = A.IdAthlete
LEFT JOIN
PARTICIPATION_EQUIPE PE ON E.IdEvenement = PE.IdEvenement
LEFT JOIN
EQUIPE EQ ON PE.IdEquipe = EQ.IdEquipe
LEFT JOIN
COMPOSITION_EQUIPE CE ON PE.IdEquipe = CE.IdEquipe
WHERE
E.IdEvenement = id_evenement
GROUP BY
PE.IdEquipe,
A.PrenomAthlete,
A.NomAthlete,
PI.NOC,
EQ.NOC,
PI.Medaille,
PE.Medaille,
PI.Resultat,
PE.Resultat
) result
) LOOP
v_resultats.EXTEND;
```


Taycir Ben Ouirane
Groupe 2E

```
49. v_resultats(v_resultats.COUNT) := rec.resultat;
50. END LOOP;
51. RETURN v_resultats;
52. END;/
```

✓ **TEST**

```
1.  DECLARE
2.  v_results t_resultats;
3.  BEGIN
4.  v_results := resultats(19005277);
5.  FOR i IN 1..v_results.COUNT LOOP
6.  DBMS_OUTPUT.PUT_LINE(v_results(i));
7.  END LOOP;
8.  END;
9.  DECLARE
10. v_results t_resultats;
11. BEGIN
12. v_results := resultats(70148);
13. FOR i IN 1..v_results.COUNT LOOP
14. DBMS_OUTPUT.PUT_LINE(v_results(i));
15. END LOOP;
16. END;/
```

❖ Procédure ajouter_resultat_individuel

```
1.  CREATE OR REPLACE PROCEDURE ajouter_resultat_individuel(
2.  id_evenement IN INTEGER,
3.  id_athlete IN INTEGER,
4.  code_noc IN VARCHAR2,
5.  resultat IN VARCHAR2
6.  )
7.  AS
8.  v_medaille VARCHAR2(10);
9.  v_count INTEGER;
10. v_current_position NUMBER;
11. v_existing_resultat VARCHAR2(10);
12. BEGIN
13. //Vérification de l'existence de l'événement
14. SELECT COUNT(*) INTO v_count FROM EVENEMENT WHERE IdEvenement =
    id_evenement;
15. IF v_count = 0 THEN
16. RAISE_APPLICATION_ERROR(-20001, 'Événement inexistant');
17. END IF;
18. //Vérification de l'existence de l'athlète
19. SELECT COUNT(*) INTO v_count FROM ATHLETE WHERE IdAthlete = id_athlete;
20. IF v_count = 0 THEN
21. RAISE_APPLICATION_ERROR(-20001, 'Athlète inexistant');
22. END IF;
23. //Vérification de l'existence du NOC
24. SELECT COUNT(*) INTO v_count FROM NOC WHERE CodeNOC = code_noc;
25. IF v_count = 0 THEN
26. RAISE_APPLICATION_ERROR(-20001, 'NOC inexistant');
27. END IF;
28. //Vérification si l'athlète a déjà un résultat pour cet événement
29. SELECT COUNT(*) INTO v_count
30. FROM PARTICIPATION_INDIVIDUELLE
31. WHERE IdEvent = id_evenement
32. AND IdAthlete = id_athlete;
33. IF v_count > 0 THEN
34. RAISE_APPLICATION_ERROR(-20003, 'Athlète déjà classé');
35. END IF;
36. //Vérification de l'unicité de la position
37. SELECT Resultat INTO v_existing_resultat
38. FROM PARTICIPATION_INDIVIDUELLE
39. WHERE IdEvent = id_evenement
```

```
AND Resultat != resultat
AND (Resultat = '1' OR Resultat = '=1');
IF v_existing_resultat IS NOT NULL THEN
RAISE_APPLICATION_ERROR(-20002, 'Position déjà occupée');
END IF;
//Vérification de l'incohérence de NOC pour cette édition des JO
SELECT COUNT(*) INTO v_count
FROM PARTICIPATION_INDIVIDUELLE PI
JOIN ATHLETE A ON PI.IdAthlete = A.IdAthlete
JOIN EVENEMENT E ON PI.IdEvent = E.IdEvenement
JOIN HOTE H ON E.IdHote = H.IdHote
WHERE E.IdEvenement = id_evenement
AND PI.NOC != code_noc
AND PI.IdAthlete = id_athlete;
v_count > 0 THEN
RAISE_APPLICATION_ERROR(-20004, 'Incohérence de NOC');
END IF;
//Détermination de la médaille
IF resultat = '1' OR resultat = '=1' THEN
v_medaille := 'Gold';
ELSIF resultat = '2' OR resultat = '=2' THEN
v_medaille := 'Silver';
ELSIF resultat = '3' OR resultat = '=3' THEN
v_medaille := 'Bronze';
END IF;
//Insertion du résultat
INSERT INTO PARTICIPATION_INDIVIDUELLE (IdEvent, IdAthlete, NOC, Resultat,
Medaille)
VALUES (id_evenement, id_athlete, code_noc, resultat, v_medaille);
COMMIT;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RAISE_APPLICATION_ERROR(-20001, 'Événement, Athlète ou NOC inexistant');
END;
/
```

✓ **Test**

```
//athlete deja classé
select idathlete from athlete where athlete.nomathlete = 'Aaltonen'and prenomathlete = 'Paavo';
BEGIN
//Appel de la procédure avec des valeurs de test
ajouter_resultat_individuel(70148, 29863, 'USA', '1');
COMMIT;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Erreur: ' || SQLERRM);
ROLLBACK;
END;
/
//athlete inexistant
select idathlete from athlete where athlete.nomathlete = 'Aaltonen'and prenomathlete = 'Paavo';
BEGIN
ajouter_resultat_individuel(70148, 2863, 'USA', '1');
COMMIT;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Erreur: ' || SQLERRM);
ROLLBACK;
END;/
```

❖ Procédure ajouter_resultat_equipe

```
CREATE OR REPLACE PROCEDURE ajouter_resultat_equipe (
id_evenement IN NUMBER,
id_equipe IN NUMBER,
resultat IN VARCHAR2
```

```
) IS
v_evenement_count NUMBER;
v_equipe_count NUMBER;
v_existing_count NUMBER;
v_existing_resultat VARCHAR2(10);
v_medaille VARCHAR2(10);
BEGIN
//Vérifier si l'événement existe
SELECT COUNT(*) INTO v_evenement_count FROM EVENEMENT WHERE IdEvenement = id_evenement;
IF v_evenement_count = 0 THEN
RAISE_APPLICATION_ERROR(-20001, 'Evenement inexistant');
END IF;
//Vérifier si l'équipe existe
SELECT COUNT(*) INTO v_equipe_count FROM EQUIPE WHERE IdEquipe = id_equipe;
IF v_equipe_count = 0 THEN
RAISE_APPLICATION_ERROR(-20001, 'Equipe inexistante');
END IF;
//Vérifier si l'équipe a déjà un résultat pour cet événement
SELECT COUNT(*), MAX(resultat) INTO v_existing_count, v_existing_resultat
FROM PARTICIPATION_EQUIPE
WHERE IdEvenement = id_evenement AND IdEquipe = id_equipe
GROUP BY IdEquipe;
IF v_existing_count > 0 AND v_existing_resultat != 'NP' THEN
RAISE_APPLICATION_ERROR(-20003, 'Equipe déjà classée');
END IF;
//Vérifier si la position est déjà occupée par une autre équipe
SELECT COUNT(*) INTO v_existing_count
FROM PARTICIPATION_EQUIPE
WHERE IdEvenement = id_evenement AND resultat = resultat AND IdEquipe != id_equipe;
IF v_existing_count > 0 THEN
RAISE_APPLICATION_ERROR(-20002, 'Position déjà occupée');
END IF;
//Déterminer la médaille en fonction du résultat
IF resultat = '1' OR resultat = '=1' THEN
v_medaille := 'Gold';
ELSIF resultat = '2' OR resultat = '=2' THEN
v_medaille := 'Silver';
ELSIF resultat = '3' OR resultat = '=3' THEN
v_medaille := 'Bronze';
ELSE
v_medaille := NULL;
END IF;
// Mettre à jour le résultat si 'NP' //est trouvé, sinon insérer un nouvel enregistrement
IF v_existing_resultat = 'NP' THEN //Mettre à jour l'enregistrement existant
UPDATE PARTICIPATION_EQUIPE
SET resultat = resultat, medaille = v_medaille
WHERE IdEvenement = id_evenement AND IdEquipe = id_equipe;
ELSE Insérer un nouvel enregistrement
INSERT INTO PARTICIPATION_EQUIPE (IdEvenement, IdEquipe, resultat, medaille)
VALUES (id_evenement, id_equipe, resultat, v_medaille);
END IF;
COMMIT;
END;/
```

✓ Test (échoué)

```
BEGIN
ajouter_resultat_equipe(905193, 19863, '=1');
END;
//vérifications
SELECT * FROM PARTICIPATION_EQUIPE WHERE IdEvenement = 905193 AND IdEquipe = 19863;
select * from participation_equipe where resultat ='NP';
```

4. Déclencheurs Imposés

❖ Création table log

```
1. CREATE TABLE LOG (
2. idLog INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
3. idAuteur INTEGER NOT NULL,
4. action VARCHAR2(20) NOT NULL,
5. dateHeureAction TIMESTAMP NOT NULL,
6. ligneAvant CLOB,
7. ligneApres CLOB
8. );
```

✓ Droits d'utilisateurs

- **Pour l'utilisateur AnalyseJO**

```
REVOKE all ON LOG FROM AnalyseJO;
```

- **Pour l'utilisateur GestionJO**

```
GRANT SELECT ON LOG TO GestionJO;
```

❖ Déclencheur Athlete

```
1. CREATE OR REPLACE TRIGGER athlete_trigger
2. AFTER INSERT OR UPDATE OR DELETE ON ATHLETE
3. FOR EACH ROW
4. DECLARE
5. v_action VARCHAR2(20);
6. BEGIN
7. IF INSERTING THEN
8. v_action := 'ajout';
9. INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
10. VALUES (:NEW.IdAthlete, v_action, SYSTIMESTAMP, NULL);
11. ELSIF UPDATING THEN
12. v_action := 'modification';
13. INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
14. VALUES (:NEW.IdAthlete, v_action, SYSTIMESTAMP, NULL, NULL);
15. ELSIF DELETING THEN
16. v_action := 'suppression';
17. INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
18. VALUES (:OLD.IdAthlete, v_action, SYSTIMESTAMP, NULL);
19. END IF;
20. END;
21. /
```

❖ Déclencheur Composition_Equipe

```
1. CREATE OR REPLACE TRIGGER composition_equipe_declencheur
2. AFTER INSERT OR UPDATE OR DELETE ON COMPOSITION_EQUIPE
3. FOR EACH ROW
4. DECLARE
5. v_action VARCHAR2(20);
6. BEGIN
7. IF INSERTING THEN
8. v_action := 'ajout';
9. INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
10. VALUES (:NEW.IdEquipe, v_action, SYSTIMESTAMP, NULL);
11. ELSIF UPDATING THEN
12. v_action := 'modification';
13. INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
14. VALUES (:NEW.IdEquipe, v_action, SYSTIMESTAMP, NULL, NULL);
15. ELSIF DELETING THEN
16. v_action := 'suppression';
17. INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
18. VALUES (:OLD.IdEquipe, v_action, SYSTIMESTAMP, NULL);
19. END IF;
20. END;
21. /
```


Taycir Ben Ouirane
Groupe 2E

❖ Déclencheur de la table Discipline

```
CREATE OR REPLACE TRIGGER discipline_trigger
AFTER INSERT OR UPDATE OR DELETE ON DISCIPLINE
FOR EACH ROW
DECLARE
v_action VARCHAR2(20);
BEGIN
IF INSERTING THEN
v_action := 'ajout';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
VALUES (:NEW.CodeDiscipline, v_action, SYSTIMESTAMP, NULL);
ELSIF UPDATING THEN
v_action := 'modification';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
VALUES (:NEW.CodeDiscipline, v_action, SYSTIMESTAMP, NULL, NULL);
ELSIF DELETING THEN
v_action := 'suppression';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
VALUES (:OLD.CodeDiscipline, v_action, SYSTIMESTAMP, NULL);
END IF;
END;
/
```

❖ Déclencheur pour la table Equipe

```
CREATE OR REPLACE TRIGGER equipe_trigger
AFTER INSERT OR UPDATE OR DELETE ON EQUIPE
FOR EACH ROW
DECLARE
v_action VARCHAR2(20);
BEGIN
IF INSERTING THEN
v_action := 'ajout';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
VALUES (:NEW.IdEquipe, v_action, SYSTIMESTAMP, NULL);
ELSIF UPDATING THEN
v_action := 'modification';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
VALUES (:NEW.IdEquipe, v_action, SYSTIMESTAMP, NULL, NULL);
ELSIF DELETING THEN
v_action := 'suppression';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
VALUES (:OLD.IdEquipe, v_action, SYSTIMESTAMP, NULL);
END IF;
END;
/
```

❖ Déclencheur pour la table Sport

```
CREATE OR REPLACE TRIGGER sport_declenheur
AFTER INSERT OR UPDATE OR DELETE ON SPORT
FOR EACH ROW
DECLARE
  v_action VARCHAR2(20);
BEGIN
  IF INSERTING THEN
    v_action := 'ajout';
    INSERT INTO LOG (idAuteur, action, dateHeureAction)
    VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP);
  ELSIF UPDATING THEN
    v_action := 'modification';
    INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
    VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP, :OLD.CodeSport || ', ' ||
:OLD.NomSport, :NEW.CodeSport || ', ' || :NEW.NomSport);
  ELSIF DELETING THEN
    v_action := 'suppression';
    INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
```


Taycir Ben Ouirane
Groupe 2E

```
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP, :OLD.CodeSport || ', ' ||
    :OLD.NomSport);
    END IF;
END;
/
```

❖ Déclencheur pour la table Evenement

```
CREATE OR REPLACE TRIGGER evenement_trigger
AFTER INSERT OR UPDATE OR DELETE ON EVENEMENT
FOR EACH ROW
DECLARE
v_action VARCHAR2(20);
BEGIN
IF INSERTING THEN
v_action := 'ajout';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
VALUES (:NEW.IdEvenement, v_action, SYSTIMESTAMP, NULL);
ELSIF UPDATING THEN
v_action := 'modification';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
VALUES (:NEW.IdEvenement, v_action, SYSTIMESTAMP, NULL, NULL);
ELSIF DELETING THEN
v_action := 'suppression';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
VALUES (:OLD.IdEvenement, v_action, SYSTIMESTAMP, NULL);
END IF;
END;
/
```

❖ Déclencheur pour la table Participation_Equipe

```
CREATE OR REPLACE TRIGGER participation_equipe_declenheur
AFTER INSERT OR UPDATE OR DELETE ON PARTICIPATION_EQUIPE
FOR EACH ROW
DECLARE
    v_action VARCHAR2(20);
BEGIN
    IF INSERTING THEN
        v_action := 'ajout';
        INSERT INTO LOG (idAuteur, action, dateHeureAction)
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP);
    ELSIF UPDATING THEN
        v_action := 'modification';
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP, :OLD.IdEquipe || ', ' ||
    :OLD.IdEvenement || ', ' || :OLD.Resultat || ', ' || :OLD.Medaille, :NEW.IdEquipe || ', ' ||
    :NEW.IdEvenement || ', ' || :NEW.Resultat || ', ' || :NEW.Medaille);
    ELSIF DELETING THEN
        v_action := 'suppression';
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP, :OLD.IdEquipe || ', ' ||
    :OLD.IdEvenement || ', ' || :OLD.Resultat || ', ' || :OLD.Medaille);
    END IF;
END;
/
```

❖ Déclencheur pour la table Participation_Individuelle

```
CREATE OR REPLACE TRIGGER participation_individuelle_declenheur
AFTER INSERT OR UPDATE OR DELETE ON PARTICIPATION_INDIVIDUELLE
FOR EACH ROW
DECLARE
    v_action VARCHAR2(20);
BEGIN
    IF INSERTING THEN
        v_action := 'ajout';
        INSERT INTO LOG (idAuteur, action, dateHeureAction)
```


Taycir Ben Ouirane
Groupe 2E

```
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP);
    ELSIF UPDATING THEN
        v_action := 'modification';
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP, :OLD.idAthlete || ', ' ||
    :OLD.idEvent || ', ' || :OLD.resultat || ', ' || :OLD.medaille || ', ' || :OLD.noc, :NEW.idAthlete || ', ' ||
    :NEW.idEvent || ', ' || :NEW.resultat || ', ' || :NEW.medaille || ', ' || :NEW.noc);
    ELSIF DELETING THEN
        v_action := 'suppression';
        INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
        VALUES ('valeur_auteur_statique', v_action, SYSTIMESTAMP, :OLD.idAthlete || ', ' ||
    :OLD.idEvent || ', ' || :OLD.resultat || ', ' || :OLD.medaille || ', ' || :OLD.noc);
    END IF;
END;
/
```

❖ Déclencheur pour la table Hote

```
CREATE OR REPLACE TRIGGER hote_trigger
AFTER INSERT OR UPDATE OR DELETE ON HOTE
FOR EACH ROW
DECLARE
v_action VARCHAR2(20);
BEGIN
IF INSERTING THEN
v_action := 'ajout';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
VALUES (:NEW.IdHote, v_action, SYSTIMESTAMP, NULL);
ELSIF UPDATING THEN
v_action := 'modification';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
VALUES (:NEW.IdHote, v_action, SYSTIMESTAMP, NULL, NULL);
ELSIF DELETING THEN
v_action := 'suppression';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
VALUES (:OLD.IdHote, v_action, SYSTIMESTAMP, NULL);
END IF;
END;
/
```

❖ Déclencheur pour la table noc

```
CREATE OR REPLACE TRIGGER noc_trigger
AFTER INSERT OR UPDATE OR DELETE ON NOC
FOR EACH ROW
DECLARE
v_action VARCHAR2(20);
BEGIN
IF INSERTING THEN
v_action := 'ajout';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneApres)
VALUES (:NEW.CodeNOC, v_action, SYSTIMESTAMP, NULL);
ELSIF UPDATING THEN
v_action := 'modification';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant, ligneApres)
VALUES (:NEW.CodeNOC, v_action, SYSTIMESTAMP, NULL, NULL);
ELSIF DELETING THEN
v_action := 'suppression';
INSERT INTO LOG (idAuteur, action, dateHeureAction, ligneAvant)
VALUES (:OLD.CodeNOC, v_action, SYSTIMESTAMP, NULL);
END IF;
END;
```

✓ Vérification des déclencheurs

```
SELECT trigger_name, table_name, trigger_type, triggering_event
FROM USER_TRIGGERS;
```


Taycir Ben Ouirane
Groupe 2E

5. Ce que vous avez fait en plus

J'ai créé une fonction get_membres_equipe(id_equipe) pour l'utiliser dans la fonction 'resultats', afin d'obtenir les membres de l'équipe participant à un évènement. Cela a permis de réduire les lignes de code et de le rendre plus clair.

```
1. CREATE OR REPLACE FUNCTION get_membres_equipe(id_equipe IN INTEGER)
2. RETURN VARCHAR2 AS
3. v_athletes VARCHAR2(4000);
4. BEGIN
5. SELECT LISTAGG(A.PrenomAthlete || ' ' || A.NomAthlete, '/') WITHIN GROUP (ORDER BY A.NomAthlete,
   A.PrenomAthlete)
6. INTO v_athletes
7. FROM COMPOSITION_EQUIPE CE
8. JOIN ATHLETE A ON CE.IdAthlete = A.IdAthlete
9. WHERE CE.IdEquipe = id_equipe;
10. RETURN v_athletes;
11. END;
12. /
```

3. Extension de la Base

Dans le cadre de ce projet, j'ai étendu la base de données existante pour inclure des informations détaillées sur les événements, de 2018 Winter Olympics, qui m'ont été attribués par le professeur. Cela comprend les événements suivants :

1. **Big Air Femmes** 9000964
2. **Simple Hommes** 400000
3. **Sprint par Équipe Hommes** 9000281

Ces événements ont été ajoutés avec leurs détails spécifiques, y compris les caractéristiques des parcours, les participants, les dates et les lieux des événements. Ci-dessous, nous présentons les dictionnaires de données pour chaque nouvelle table ainsi que les scripts SQL pour créer ces tables :

6. Big Air Femmes

Dictionnaire de données :

Colonne	Description	Type
Date_Evenement	Date de l'événement	A
IdEvenement	Identifiant de l'événement	A
Statut	Statut de l'événement	A
Lieu	Lieu de l'événement	A
Participants	Nombre de participants	A
Longueur_Rampe	Longueur de la rampe	A
Pente_Rampe	Pente de la rampe	A
Hauteur_Saut	Hauteur du saut	A
Pente_Atterrissage	Distance de décollage jusqu'au Knoll Pente d'atterrissage	A

Script de création de la table :

```
1.  CREATE TABLE Big_Air_Femmes (
2.    IdEvenement NUMBER PRIMARY KEY,
3.    Date_Evenement DATE NOT NULL,
4.    Statut VARCHAR2(50),
5.    Lieu VARCHAR2(100),
6.    Participants INTEGER,
7.    Longueur_Rampe INTEGER,
8.    Pente_Rampe INTEGER,
9.    Hauteur_Saut INTEGER,
10.   Pente_Atterrissage INTEGER,
11.   Distance_Decollage_Knoll INTEGER,
12.   CONSTRAINT fk_BigAirFemmes_Evenement FOREIGN KEY (IdEvenement) REFERENCES
      EVENEMENT(IdEvenement)
```

Droits d'utilisateurs

```
1.  GRANT SELECT, INSERT, UPDATE, DELETE ON Big_Air_Femmes TO gestionJO;
2.  GRANT SELECT ON Big_Air_Femmes TO analyseJO;
```

7. Simple Hommes

Dictionnaire de données :

Colonne	Description	Type
Date_Evenement	Date de l'événement	A
IdEvenement	Identifiant de l'événement	A
Statut	Statut de l'événement	A
Lieu	Lieu de l'événement	A
Participants	Nombre de participants	A
Type_Jeu	Type de jeu	A
Format	Format de l'événement	A
Curves	Nombre de virages	A
Length	Longueur du parcours	A
Start_Altitude	Altitude de départ	A
Vertical_Drop	Dénivelé vertical	A

Script de création de la table :

```
1.  CREATE TABLE Singles_Men (
2.  IdEvenement NUMBER,
3.  Date_Evenement DATE,
4.  Statut VARCHAR2(50),
5.  Lieu VARCHAR2(100),
6.  Participants INTEGER,
7.  Format VARCHAR2(100),
8.  Curves INTEGER,
9.  Length DECIMAL(7,2),
10. Start_Altitude INTEGER,
11. Vertical_Drop INTEGER,
12. PRIMARY KEY (IdEvenement, Date_Evenement),
13. CONSTRAINT fk_SinglesMen_Evenement FOREIGN KEY (IdEvenement) REFERENCES
    EVENEMENT(IdEvenement)
```

```
14. );
```

Droits d'utilisateurs

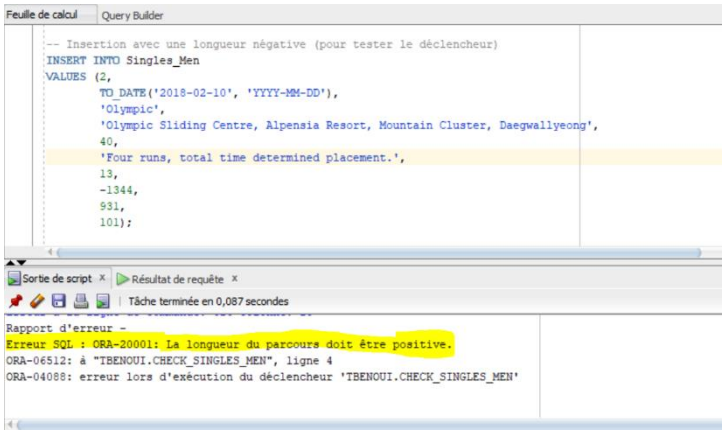
```
3. GRANT SELECT, INSERT, UPDATE, DELETE ON Singles_Men TO gestionJO;
4. GRANT SELECT ON Singles_Men TO analyseJO;
```

Déclencheur

```
1. CREATE OR REPLACE TRIGGER check_singles_men
2. BEFORE INSERT OR UPDATE ON Singles_Men
3. FOR EACH ROW
4. BEGIN
5. //Vérifier que la longueur du parcours est positive
6. IF :NEW.Length <= 0 THEN
7. RAISE_APPLICATION_ERROR(-20001, 'La longueur du parcours doit être positive.');
8. END IF;
9. END;
10. /
```

✓ **Test**

```
1. INSERT INTO Singles_Men (IdEvenement, Date_Evenement, Statut, Lieu, Participants, Format, Curves,
   Length, Start_Altitude, Vertical_Drop)
VALUES (2, TO_DATE('2018-02-10', 'YYYY-MM-DD'), 'Olympic', 'Olympic Sliding Centre, Alpensia Resort, Mountain
Cluster, Daegwallyeong', 40, 'Four runs, total time determined placement.', 13, -1344, 931, 101);
```



8. Sprint par Équipe Hommes

Dictionnaire de données :

Colonne	Description	Type
Date_Evenement	Date de l'événement	A
IdEvenement	Identifiant de l'événement	A
Statut	Statut de l'événement	A
Lieu	Lieu de l'événement	A
Participants	Nombre de participants	A
Course_Length	Longueur du parcours	A
Height_Differential	Différence de hauteur	A

Maximum_Climb	Montée maximale	A
Total_Climbing	Ascension totale	A

Script de création de la table :

```
1.  CREATE TABLE Team_Sprint_Men (
2.     IdEvenement NUMBER,
3.     Date_Evenement DATE,
4.     Statut VARCHAR2(50),
5.     Lieu VARCHAR2(100),
6.     Participants INTEGER,
7.     Format VARCHAR2(100),
8.     Course_Length DECIMAL(7,2),
9.     Height_Differential INTEGER,
10.    Maximum_Climb INTEGER,
11.    Total_Climbing INTEGER,
12.    PRIMARY KEY (IdEvenement, Date_Evenement),
13.    CONSTRAINT fk_TeamSprintMen_Evenement FOREIGN KEY (IdEvenement) REFERENCES
       EVENEMENT(IdEvenement)
14. );
```

En résumé, ces extensions améliorent la base de données en permettant la prise en charge des détails spécifiques des événements qui m'ont été attribués, tout en maintenant la cohérence et l'intégrité des données. Ces propositions seront améliorées pendant la deuxième partie du projet. Des fonctions PL/SQL seront mises en place.

9. Exemple d'insertion

Voici un exemple de requête d'insertion pour chaque nouvelle table :

Big Air Femmes

```
1. INSERT INTO Big_Air_Femmes (Date_Evénement, Statut, Lieu, Participants, Longueur_Rampe,
   Pente_Rampe, Hauteur_Saut, Pente_Atterrissage, Distance_Décollage_Knoll)
2. VALUES ('2018-02-19', 'Olympique', 'Phoenix Snow Park, Mountain Cluster, Bongpyeong', 26, 73, 39, 4.9,
   38, 19.5);
```

Simple Hommes

```
1. INSERT INTO Singles_Men (IdEvenement, Date_Evenement, Statut, Lieu, Participants, Format, Curves,
   Length, Start_Altitude, Vertical_Drop)
2. VALUES (1, TO_DATE('2018-02-10', 'YYYY-MM-DD'), 'Olympic', 'Olympic Sliding Centre, Alpensia
   Resort, Mountain Cluster, Daegwallyeong', 40, 'Four runs, total time determined placement.', 13, 1344, 931,
   101);
```

Sprint par Équipe Hommes

```
1. INSERT INTO Team_Sprint_Men (IdEvenement, Date_Evenement, Statut, Lieu, Participants, Format,
   Course_Length, Height_Differential, Maximum_Climb, Total_Climbing)
2. VALUES (1, TO_DATE('2018-02-21', 'YYYY-MM-DD'), 'Olympic', 'Alpensia Cross-Country Centre,
   Alpensia Resort, Mountain Cluster, Daegwallyeong', 56, 'Two-man teams, each skiing three alternate legs of
   approximately 1,500 metres.', 1381, 27, 25, 53) ;
```

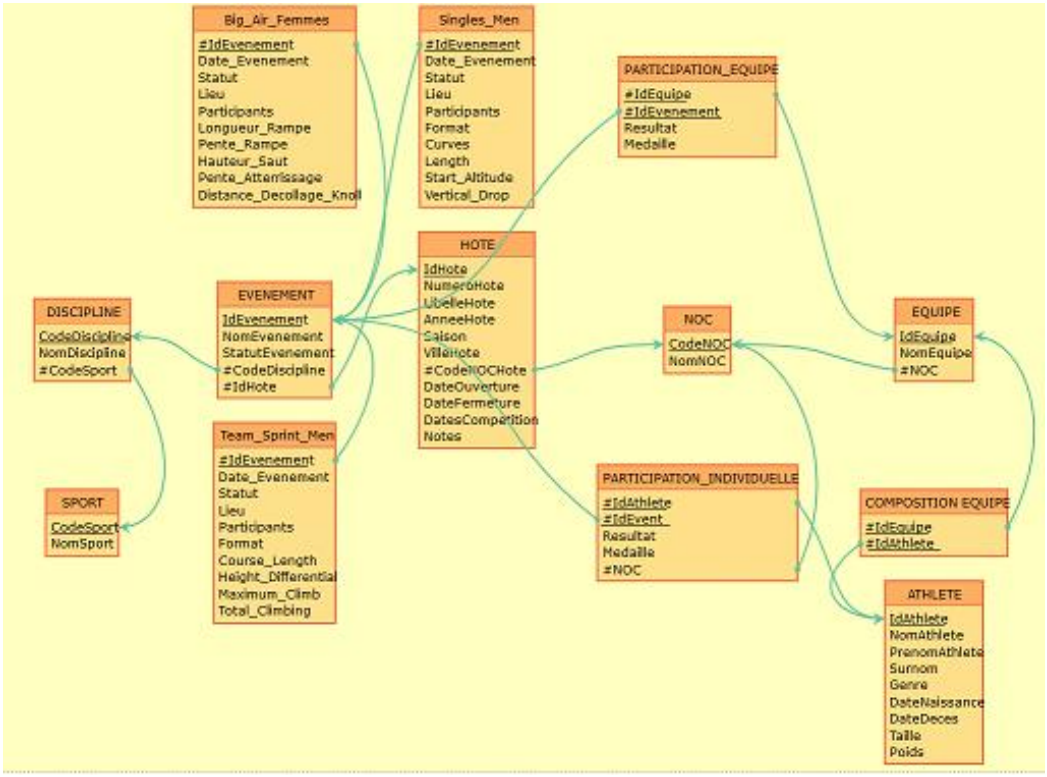

Taycir Ben Ouirane
Groupe 2E

10. Fonctions et procédures

- Cette fonction prend un identifiant d'événement en entrée et renvoie un curseur avec les détails de l'événement correspondant.

```
1. CREATE OR REPLACE FUNCTION obtenir_details_evenement(id_evenement IN INTEGER)
2. RETURN sys_refcursor AS
3. curseur_details_evenement sys_refcursor;
4. BEGIN
5. OPEN curseur_details_evenement FOR
6. SELECT *
7. FROM EVENEMENT
8. WHERE IdEvenement = id_evenement;
9. RETURN curseur_details_evenement;
10. END;
11. /
```

11. Schéma relationnel



4. Conclusion

En conclusion, ce rapport présente les différentes étapes suivies pour étendre la base de données. Par ailleurs, j'ai proposé des modifications permettant d'améliorer la gestion et l'exploitation de la base de données, afin de prendre en charge les résultats détaillés des événements sportifs Big Air Femmes, Simple Hommes, et Sprint par Équipe Hommes. D'autres fonctions peuvent être mises en place pour mieux gérer les données spécifiques à ces événements. Ces améliorations seront développées pendant la 2ème partie du projet.

Taycir Ben Ouirane
Groupe 2E

5. Mon bilan

Les droits d'utilisateurs	Les bons privilèges ont été bien données aux deux utilisateurs.
Vues (medailles_athlete, medailles_noc)	Tout va bien.
Fonction biographie	La fonction renvoie bien les données demandées. Cependant le JSON renvoyé n'est pas le bon. Les utilisateurs disposent du droit d'exécution.
Fonction résultats	La fonction récupère les bonnes données, on voit le bon résultat affiché. Cependant, ce n'est pas sous forme d'un tableau à 3 colonnes.
Procédure ajout_resultat_equipe	La procédure compile, traite les exceptions demandées. Cependant, ne fonctionne pas bien… problème non détecté.
Procédure ajout_resultat_individuel	La procédure compile, traite les exceptions demandées. Cependant, ne fonctionne pas bien… problème non détecté.
Table Log	Tout va bien.
Les déclencheurs	Le code me semble bon.

✓ **Compétences renforcées**

- Importation de données des données à partir de fichiers CSV
- Attribution de droits aux utilisateurs
- Création des vues, fonctions et procédures qui facilitent l'analyse des données
- Développement de déclencheurs

Taycir Ben Ouirane
Groupe 2E