CENTRALE LYON

Systèmes de gestion de bases de données

Basketball

Élèves :
Baptiste Campeas
Philomène Carrel

Enseignants :
Mohsen Ardabilian

5 janvier 2026

Table des matières

1 Conception du modèle de données

Cette section présente la modélisation réalisée pour le Projet 3 "Basketball". Conformément aux consignes permettant le choix des outils d'implémentation, nous avons opté pour une conception du schéma relationnel assistée par des scripts Python pour la génération du SQL (DDL et DML), privilégiant cette approche à l'utilisation du générateur automatique DB-MAIN.

1.1 Diagramme Relationnel

Le diagramme ci-dessous illustre la structure finale de la base de données. Il met en évidence les tables, les clés primaires et étrangères, ainsi que les relations de cardinalité.

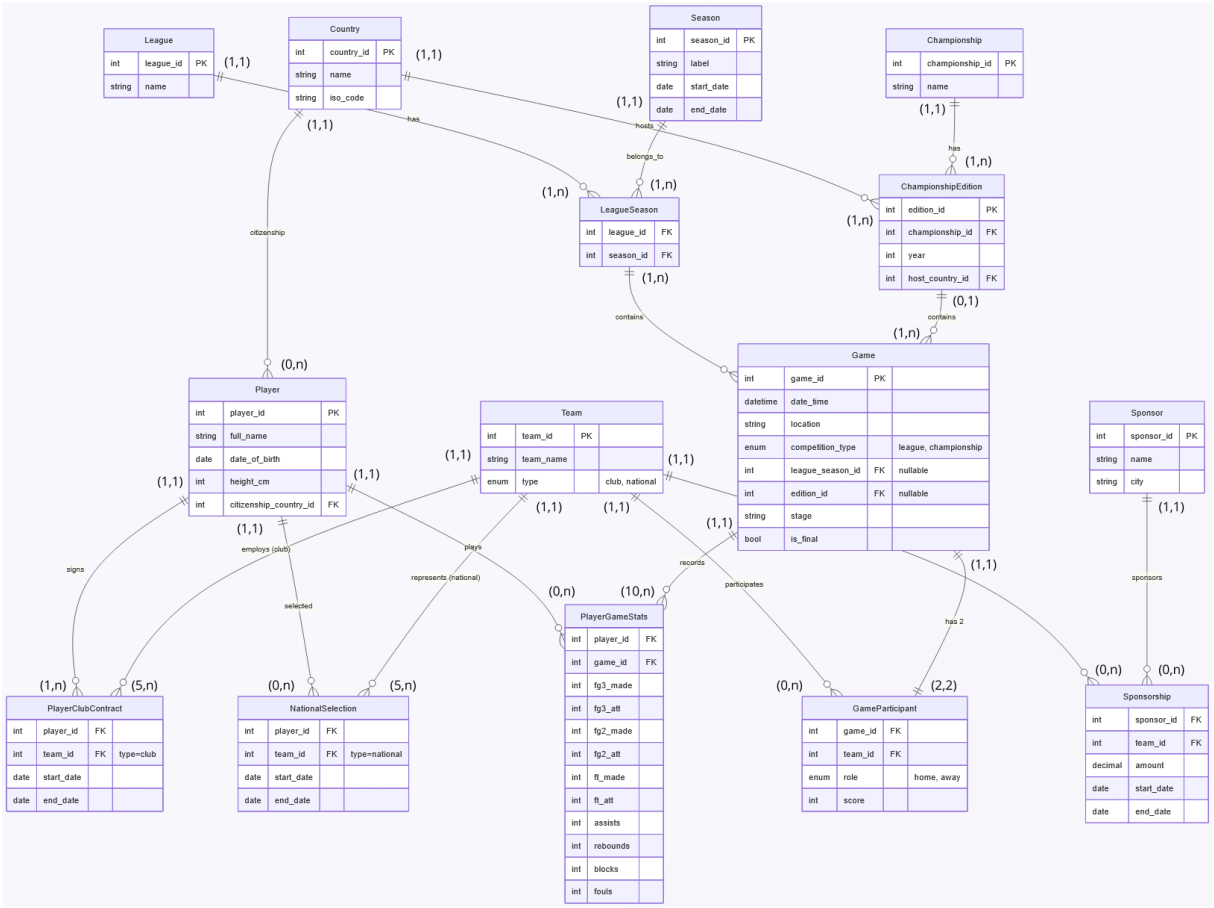

FIGURE 1 – Schéma Relationnel de la base de données Basketball

1.2 Définition textuelle des entités principales

Le modèle s'articule autour des entités suivantes :

— **Player (Joueur)** : Identifié par un ID unique, caractérisé par ses données anthropométriques et sa nationalité.

— **Team (Équipe) :** Nous avons hésité entre créer deux tables distinctes (`Club` et `NationalTeam`) ou une table unique. Nous avons finalement opté pour une stratégie d'héritage par table unique (Single Table Inheritance) avec un attribut discriminant

`type`. Ce choix simplifie les requêtes de jointure pour les matchs, car une table `Game` peut référencer une `Team` sans se soucier de son type.
— **Game (Match)** : Événement central reliant deux équipes, situé dans le temps (Saison/Édition) et l'espace (Lieu).
— **PlayerGameStats** : Table d'association porteuse de données, stockant la performance fine (points, rebonds, fautes) d'un joueur pour un match donné.

1.3 Justification de la Normalisation

Le schéma relationnel proposé a été conçu pour respecter les formes normales afin d'éviter la redondance et les anomalies de mise à jour.

Première Forme Normale (1FN)

Nous avons travaillé à éliminer tout attribut multivalué pour garantir l'atomicité des données. Le cas concret s'est posé pour la gestion des sponsors : un club pouvant en avoir plusieurs, nous avons écarté l'idée de stocker une liste (ex : "Nike, Adidas") dans la table `Team`. Nous avons privilégié la création d'une table de liaison dédiée `Sponsorship`, assurant ainsi qu'une case ne contient jamais qu'une seule valeur.

Deuxième Forme Normale (2FN)

Notre attention s'est portée sur les tables possédant une clé primaire composée, comme `PlayerGameStats` (identifiée par le couple `player_id` + `game_id`). Nous avons vérifié que chaque statistique (points, rebonds, fautes) dépendait bien de la combinaison "Joueur ET Match". Si nous avions inclus la taille du joueur dans cette table, cela aurait violé la 2FN car la taille ne dépend que du joueur ; cette information est donc correctement restée isolée dans la table `Player`.

Troisième Forme Normale (3FN)

Nous avons veillé à respecter la 3ème Forme Normale (3FN). Le point de vigilance principal concernait les joueurs et leur nationalité : plutôt que de stocker le nom du pays dans la table `Player` (ce qui créerait une redondance), nous passons par une clé étrangère `citizenship_country_id`. De même, les statistiques de match dépendent bien de la clé composite (Joueur + Match), validant la 2FN.

1.4 Contraintes d'intégrité

L'intégrité des données est assurée par le moteur de base de données via :
— **Intégrité Référentielle :** Toutes les relations (FK) sont contraintes. Impossible de supprimer un `Country` s'il est référencé par un `Player`.
— **Héritage et Types :** La distinction entre Club et Équipe Nationale est gérée par une contrainte de domaine sur l'attribut `team.type` et par la logique applicative lors de l'inscription aux compétitions (`League` vs `Championship`).
— **Unicité temporelle :** Un joueur ne peut avoir qu'un seul contrat actif (`PlayerClubContract`) à une date, contrainte qui sera vérifiée par triggers ou par l'application Python lors de l'insertion.

2 Réalisation et Implémentation

Cette section détaille la traduction du modèle conceptuel en une base de données opérationnelle SQLite, ainsi que les requêtes développées pour l'interface de gestion.

2.1 Tables Relationnelles

Le schéma relationnel est constitué des tables suivantes, normalisées pour éviter la redondance :

Tables de Référence : `Country` (Pays), `League` (Ligues), `Season` (Saisons), `Championship` (Championnats), `Sponsor`.

Acteurs :
- `Player` : Contient les informations stables des joueurs (Nom, Taille, Nationalité).
- `Team` : Table unique regroupant Clubs et Équipes Nationales, différenciés par l'attribut `type`.

Relations Temporelles :
- `PlayerClubContract` : Historique des contrats en club.
- `NationalSelection` : Historique des sélections nationales.
- `Sponsorship` : Historique des contrats de sponsoring.

Compétitions et Matchs :
- `Game` : Table centrale des matchs. Elle contient une contrainte de validation (CHECK) pour assurer qu'un match appartient soit à une ligue, soit à une édition de championnat, mais pas aux deux simultanément.
- `GameParticipant` : Associe deux équipes à un match (Domicile/Extérieur) et stocke le score.
- `PlayerGameStats` : Table de faits contenant les performances détaillées (3 points, rebonds, fautes, etc.) pour chaque joueur par match.

2.2 Script de Création et Contraintes

La base de données est initialisée via un script SQL. Nous utilisons des **Triggers** pour assurer l'intégrité métier complexe que les contraintes simples (CHECK) ne peuvent couvrir (ex : vérifier le type d'équipe).

```
-- Exemple de la table Team avec contrainte de type
CREATE TABLE Team (
  team_id   INTEGER PRIMARY KEY,
  team_name TEXT NOT NULL,
  type      TEXT NOT NULL CHECK (type IN ('club','national'))
);

-- Table des contrats joueurs avec clés étrangères
CREATE TABLE PlayerClubContract (
  player_id  INTEGER NOT NULL,
  team_id    INTEGER NOT NULL,
  start_date TEXT    NOT NULL,
  end_date   TEXT,
  PRIMARY KEY (player_id, team_id, start_date),
  FOREIGN KEY (player_id) REFERENCES Player(player_id),
```

```
  FOREIGN KEY (team_id)   REFERENCES Team(team_id)
);

-- Trigger assurant qu'un contrat 'Club' ne lie pas une équipe 'Nationale'
CREATE TRIGGER trg_pcc_team_is_club_ins
BEFORE INSERT ON PlayerClubContract
FOR EACH ROW
WHEN (SELECT type FROM Team WHERE team_id = NEW.team_id) <> 'club'
BEGIN
  SELECT RAISE(ABORT, 'PlayerClubContract requires a club team');
END;
```

2.3 Gestion des Vues

Dans notre architecture applicative (Flask + SQLAlchemy), nous avons fait le choix de ne pas stocker de vues statiques (`CREATE VIEW`) dans la base de données.

Les vues sont gérées dynamiquement par la couche applicative. Cela permet de paramétrer les requêtes (ex : changer l'année ou l'ID du club à la volée) plus efficacement qu'avec des vues SQL statiques. Les requêtes complexes présentées ci-dessous agissent comme des vues logiques pour l'utilisateur final.

2.4 Requêtes d'accès aux données (Requêtes Métier)

Les requêtes suivantes correspondent aux exigences du cahier des charges. Elles sont implémentées en SQL natif via l'application Python.

1. Top 10 des joueurs (Points totaux en équipe nationale)

Cette requête réalise une jointure entre les joueurs, leurs stats, et filtre uniquement sur les matchs joués sous le maillot national.

```
SELECT p.full_name, t.team_name,
       SUM(pgs.ft_made + 2*pgs.fg2_made + 3*pgs.fg3_made) as total_points
FROM Player p
JOIN PlayerGameStats pgs ON p.player_id = pgs.player_id
JOIN GameParticipant gp ON pgs.game_id = gp.game_id
JOIN Team t ON gp.team_id = t.team_id
WHERE t.type = 'national'
  AND pgs.player_id IN (
      SELECT ns.player_id FROM NationalSelection ns WHERE ns.team_id = t.team_id
  )
GROUP BY p.player_id
ORDER BY total_points DESC
LIMIT 10;
```

2. Top 3 adresse aux lancers-francs (Finale Euro 2002)

Calcul du pourcentage de réussite lors d'un événement spécifique.

```
SELECT p.full_name,
       (CAST(pgs.ft_made AS FLOAT) / pgs.ft_att) * 100 as ft_percent
FROM Player p
JOIN PlayerGameStats pgs ON p.player_id = pgs.player_id
JOIN Game g ON pgs.game_id = g.game_id
JOIN ChampionshipEdition ce ON g.edition_id = ce.edition_id
WHERE ce.championship_id = 1
  AND ce.year = 2002
  AND g.stage = 'Final'
  AND pgs.ft_att > 0
ORDER BY ft_percent DESC
LIMIT 3;
```

3. Club ayant la plus grande moyenne de taille

Agrégation par club sur la table des joueurs sous contrat.

```
SELECT t.team_name, AVG(p.height_cm) as avg_height
FROM Team t
JOIN PlayerClubContract pcc ON t.team_id = pcc.team_id
JOIN Player p ON pcc.player_id = p.player_id
WHERE t.type = 'club'
GROUP BY t.team_id
ORDER BY avg_height DESC
LIMIT 1;
```

4. Sponsor des champions du monde

Requête complexe impliquant 6 jointures pour relier le sponsor au résultat du match final (victoire).

```
SELECT s.name, COUNT(DISTINCT ce.year) as titles_count
FROM Sponsor s
JOIN Sponsorship sp ON s.sponsor_id = sp.sponsor_id
JOIN Team t ON sp.team_id = t.team_id
JOIN GameParticipant gp ON t.team_id = gp.team_id
JOIN Game g ON gp.game_id = g.game_id
JOIN ChampionshipEdition ce ON g.edition_id = ce.edition_id
WHERE ce.championship_id = 2  -- ID World Championship
  AND g.is_final = 1
  -- Le score de l'équipe est le score MAX du match (donc victoire)
  AND gp.score = (SELECT MAX(score) FROM GameParticipant WHERE game_id = g.game_id)
GROUP BY s.sponsor_id
ORDER BY titles_count DESC;
```

5. Meilleur tireur à 3 points par club (Saison courante)

Cette requête a été optimisée pour effectuer l'intégralité du traitement côté serveur SQL (approche "Full SQL"). Contrairement à une approche où l'application filtrerait les

données, nous utilisons ici une CTE pour calculer les statistiques, couplée à une fonction de fenêtrage (`RANK() OVER`) pour classer les joueurs au sein de leur équipe.

```
WITH PlayerStats AS (
    SELECT
        t.team_name,
        p.full_name,
        (CAST(SUM(pgs.fg3_made) AS FLOAT) / SUM(pgs.fg3_att)) * 100 as pct
    FROM PlayerGameStats pgs
    JOIN Game g ON pgs.game_id = g.game_id
    JOIN GameParticipant gp ON g.game_id = gp.game_id
    JOIN Team t ON gp.team_id = t.team_id
    JOIN Player p ON pgs.player_id = p.player_id
    WHERE g.competition_type = 'league'
      AND g.league_season_id_season = 1 -- Saison courante
      AND t.type = 'club'
      AND gp.team_id IN (
          SELECT team_id FROM PlayerClubContract
          WHERE player_id = pgs.player_id
      )
    GROUP BY t.team_id, p.player_id
    HAVING SUM(pgs.fg3_att) > 0
),
RankedStats AS (
    SELECT
        team_name,
        full_name,
        pct,
        -- Utilisation de 'rnk' pour éviter le mot-clé réservé 'RANK'
        RANK() OVER (PARTITION BY team_name ORDER BY pct DESC) as rnk
    FROM PlayerStats
)
SELECT team_name, full_name, pct
FROM RankedStats
WHERE rnk = 1;
```

6. Meilleur passeur d'un club donné

```
SELECT p.full_name, AVG(pgs.assists) as avg_assists
FROM PlayerGameStats pgs
JOIN Game g ON pgs.game_id = g.game_id
JOIN GameParticipant gp ON g.game_id = gp.game_id
JOIN Player p ON pgs.player_id = p.player_id
WHERE gp.team_id = :club_id -- Paramètre injecté (ex: 4 pour Real Madrid)
GROUP BY p.player_id
ORDER BY avg_assists DESC
LIMIT 1;
```

7. Clubs titrés plus de 3 fois en Euroligue

Utilisation de la clause `HAVING` pour filtrer sur le résultat de l'agrégation.

```
SELECT t.team_name, COUNT(ce.year) as wins
FROM Team t
JOIN GameParticipant gp ON t.team_id = gp.team_id
JOIN Game g ON gp.game_id = g.game_id
JOIN ChampionshipEdition ce ON g.edition_id = ce.edition_id
WHERE ce.championship_id = 3 -- ID Euroleague
  AND g.is_final = 1
  AND gp.score = (SELECT MAX(score) FROM GameParticipant WHERE game_id = g.game_id)
GROUP BY t.team_id
HAVING wins > 3;
```

3 Interface Utilisateur et Architecture Applicative

Au-delà des requêtes SQL, notre objectif était de rendre cette base de données utilisable par un non-spécialiste. Nous avons donc développé une interface web, avec une ergonomie moderne et colorée.

3.1 Architecture : Flask

Nous avons choisi le micro-framework Flask pour sa légèreté. Flask nous a obligés à structurer nous-mêmes notre architecture MVC (Modèle-Vue-Contrôleur).

- **Côté Données :** L'utilisation de SQLAlchemy nous a permis de manipuler nos tables `Player` ou `Game` directement sous forme d'objets Python, sécurisant les interactions.
- **Côté Rendu :** Le moteur de template Jinja2 assure la génération dynamique des pages HTML, en injectant les données SQL directement dans le balisage.

3.2 Choix Esthétiques et Expérience Utilisateur

Nous ne voulions pas de simples tableaux HTML bruts. Nous avons travaillé sur une identité visuelle propre pour rendre la navigation agréable :

- **Identité Visuelle :** Nous avons opté pour une palette de couleurs violet-bleu (*Linear Gradient*), rappelant l'univers sportif.
- **Design :** Pour aérer l'information, les données ne sont pas plaquées sur le fond mais contenues dans des "cartes" blanches avec des ombres portées (*Box Shadow*), créant un effet de profondeur et de hiérarchie visuelle.
- **Indicateurs Visuels :** Afin de faciliter la lecture des listes de matchs, nous avons implémenté des indicateurs visuels conditionnels. Par exemple, comme visible sur la capture d'écran, les matchs de finale sont automatiquement marqués par un badge doré "Trophée", permettant de repérer les événements majeurs d'un simple coup d'œil.

3.3 Aperçu du résultat

La figure ci-dessous illustre la liste des matchs. On y retrouve nos choix de design : le fond dégradé, la structure en cartes, et la mise en évidence des finales.

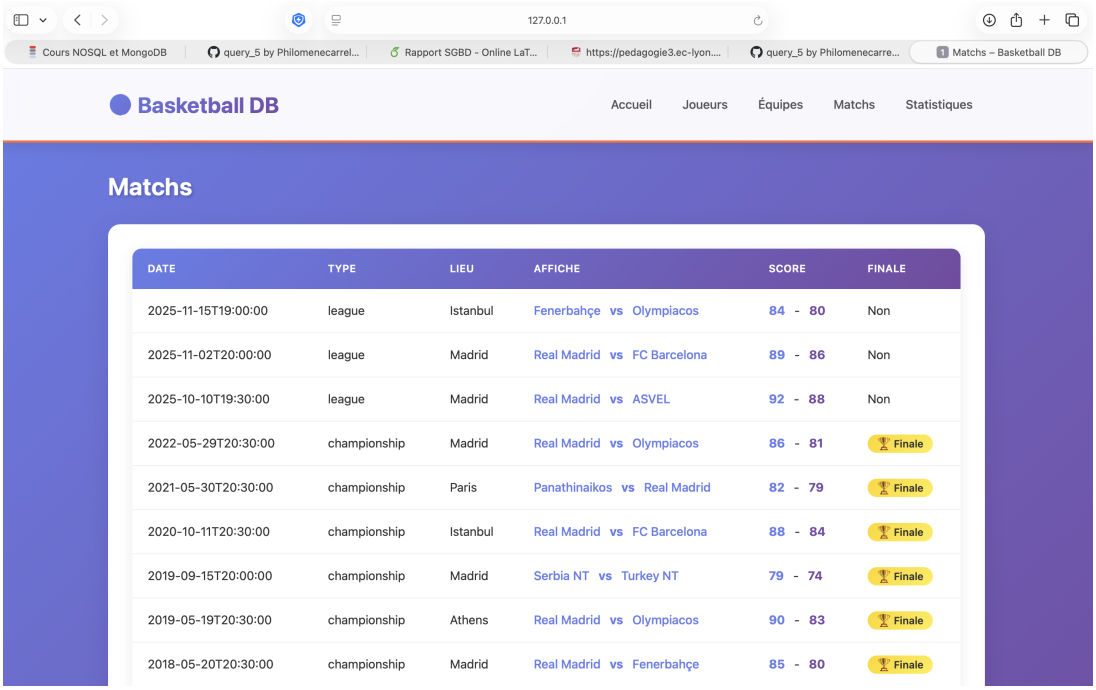

FIGURE 2 – Capture d'écran de l'interface web développée avec Flask

4 Conclusion et Perspectives

La réalisation de ce projet "Basketball" a permis de confronter les concepts théoriques de modélisation à la réalité de l'implémentation applicative, à travers une démarche structurée en trois phases distinctes.

La phase initiale de **conception conceptuelle** a d'abord imposé une rigueur structurelle nécessaire pour traduire des règles de gestion complexes (héritage des équipes, historicisation des contrats) en un schéma normalisé. Cette étape s'est prolongée par l'**implémentation physique** en SQL, où la définition fine des contraintes d'intégrité et des déclencheurs (*triggers*) a permis de verrouiller la cohérence des données à la source. Enfin, l'**intégration applicative** sous Flask a mis en lumière les enjeux d'interopérabilité et de performance, soulignant la nécessité d'arbitrer entre traitement logique côté serveur et optimisation des requêtes via le SGBD.

Limites et Améliorations futures

Bien que le système réponde à l'ensemble du cahier des charges, il présente certaines limites inhérentes au temps imparti comme l'absence de gestion des droits utilisateurs. Une évolution du projet consisterait à implémenter un contrôle d'accès pour distinguer les simples visiteurs des administrateurs pouvant modifier les données.

En définitive, ce projet valide notre capacité à construire une chaîne complète, allant d'un schéma relationnel normalisé jusqu'à sa mise en production au sein d'une architecture MVC.