Modélisation du tournoi de Roland Garros

SERY Ange

Contents

1 Cahier des charges **2**

2 Tout dans une table **2**
2.1 Modélisation 2
2.2 Structuration 2
2.3 Avantages et limites 3

3 Modélisation **3**
3.1 Entité-Association 3
3.2 Associations entre les entités 4
3.2.1 Association Joueur-Tournoi 4
3.2.2 Association Joueur-Match 4
3.2.3 Association Match-Tournoi 4
3.2.4 Association Joueur-Score 4
3.2.5 Association Match-Score 4
3.2.6 Association Match-Gagnant 4
3.2.7 Association Tournoi-Catégorie 5
3.2.8 Association Catégorie-Gain 5
3.3 Structuration des tables 5
3.4 Création de la base 6
3.5 Insertion des informations 6
3.6 État de la base de données 7

4 Cas d'Usages **7**
4.1 Recherche d'Informations 7
4.1.1 Exemple 1 : Liste des joueurs d'un tournoi donné 7
4.1.2 Exemple 2 : Trouver le vainqueur d'un match spécifique 7
4.1.3 Exemple 3 : Scores détaillés d'un match donné 8
4.1.4 Exemple 4 : Nombre total de sets joués dans un tournoi 8
4.1.5 Exemple 5 : Gains totaux par joueur 8
4.1.6 Exemple 6 : Classement des joueurs par leur nationalité 9
4.1.7 Exemple 7 : Liste des joueurs sélectionnés pour la finale 9

5 Mises à jour par des utilisateurs et par programmation **9**
5.1 De la table au tableur 9
5.2 Du tableur à la table 10
5.3 Calcul des scores 11
5.4 Applications pratiques 12

6 Conclusion **12**

1 Cahier des charges

L'objectif de ce projet est de modéliser le tournoi de Roland Garros afin de permettre l'extraction de statistiques. Le cahier des charges défini est le suivant :

- Le tournoi comprend plusieurs catégories : **hommes**, **femmes**, **juniors**, etc.
- Chaque catégorie regroupe des joueurs qui participent à des **matchs à élimination directe**.
- Les informations des joueurs incluent : *nom*, *prénom*, *nationalité*, *sponsor principal* et *classement ATP*.
- Chaque **match** oppose deux joueurs d'une catégorie donnée et d'un **tour** spécifique (1er tour, finale, etc.).
- Un match se compose d'un nombre de **sets**, chaque set comprenant un nombre de **jeux** marqués par chaque joueur.
- Les **gains** d'un joueur sont fonction de sa catégorie et de son niveau d'avancement dans le tournoi.

2 Tout dans une table

Dans une première approche simplifiée, toutes les informations du tournoi seront regroupées dans une seule table appelée **matchs**. Cela permettra de manipuler directement les données du tournoi.

2.1 Modélisation

L'entité unique est **Matchs**. Elle inclut les informations suivantes :

- **match_id** : Identifiant unique du match.
- **categorie** : Catégorie du tournoi (Homme, Femme, Junior, etc.).
- **tour** : Niveau du tournoi (1er tour, 2e tour, finale...).
- **joueur_1**, **joueur_2** : Identifiants des deux joueurs impliqués.
- **sets_joueur_1**, **sets_joueur_2** : Nombre de sets gagnés par chaque joueur.
- **jeux_joueur_1**, **jeux_joueur_2** : Détail des jeux par set.
- **gain_joueur_1**, **gain_joueur_2** : Gains financiers des joueurs.

2.2 Structuration

Voici la structure de la table en SQL :

```sql
CREATE TABLE Tournoi (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    categorie VARCHAR(50) NOT NULL,
    nom_joueur1 VARCHAR(100) NOT NULL,
    prenom_joueur1 VARCHAR(100) NOT NULL,
```

```
    nationalite_joueur1 VARCHAR(50),
    sponsor_joueur1 VARCHAR(100),
    classement_atp_joueur1 INT,
    nom_joueur2 VARCHAR(100) NOT NULL,
    prenom_joueur2 VARCHAR(100) NOT NULL,
    nationalite_joueur2 VARCHAR(50),
    sponsor_joueur2 VARCHAR(100),
    classement_atp_joueur2 INT,
    tour VARCHAR(50) NOT NULL,
    niveau_tour INT NOT NULL,
    set1_jeux_joueur1 INT,
    set1_jeux_joueur2 INT,
    set2_jeux_joueur1 INT,
    set2_jeux_joueur2 INT,
    set3_jeux_joueur1 INT,
    set3_jeux_joueur2 INT,
    set4_jeux_joueur1 INT,
    set4_jeux_joueur2 INT,
    set5_jeux_joueur1 INT,
    set5_jeux_joueur2 INT,
    gagnant VARCHAR(100),
    gain DECIMAL(10, 2),
    nombre_sets INT NOT NULL
);
```

2.3 Avantages et limites

Avantages :

- Simplification de la structure pour un prototypage rapide.
- Accès direct aux données des matchs pour des statistiques simples.

Limites :

- Répétition des informations (catégorie, gains...).
- Difficulté à gérer les relations complexes (joueurs, tournois).

3 Modélisation

Pour une structure plus robuste, les informations seront réparties dans plusieurs tables interconnectées selon le modèle Entité-Association.

3.1 Entité-Association

Les entités principales et leurs relations sont identifiées comme suit :

- **Catégorie** : Représente les groupes de joueurs.
- **Joueur** : Contient les informations personnelles des joueurs.
- **Tournoi** : Représente une édition spécifique avec ses dates.

- **Match** : Représente une rencontre entre deux joueurs.
- **Score** : Contient les résultats des sets pour chaque match.
- **Gain** : Définit les montants des gains selon la catégorie et le tour.

3.2 Associations entre les entités

Les relations entre les différentes entités de notre projet de gestion de tournoi de tennis sont décrites ci-dessous. Chaque association est présentée sous la forme **Sujet-Verbe-Complément**, accompagnée des cardinalités correspondantes.

3.2.1 Association Joueur-Tournoi

- **Une joueur participe à des tournois** : Un joueur peut participer à 0 ou plusieurs tournois.
- **Un tournoi sélectionne des joueurs** : Un tournoi implique plusieurs joueurs (au moins deux).

3.2.2 Association Joueur-Match

- **Un joueur participe à des matchs** : Un joueur peut participer à 0 ou plusieurs matchs.
- **Un match oppose des joueurs** : Un match oppose exactement deux joueurs.

3.2.3 Association Match-Tournoi

- **Un match appartient à un tournoi** : Un match appartient à un seul tournoi.
- **Un tournoi est constitué de matchs** : Un tournoi inclut plusieurs matchs (au moins un).

3.2.4 Association Joueur-Score

- **Un joueur obtient un score** : Un joueur peut obtenir un ou plusieurs scores dans le cadre des sets d'un match.
- **Un score est attribué à un joueur** : Un score appartient à un seul joueur.

3.2.5 Association Match-Score

- **Un match a un score global** : Un match peut produire 0 ou plusieurs scores pour représenter les sets.
- **Un score est rattaché à un match** : Un score est associé à un seul match.

3.2.6 Association Match-Gagnant

- **Un match désigne un gagnant** : Un match désigne un seul joueur gagnant ou aucun si le match est annulé.
- **Un gagnant est désigné pour un match** : Un gagnant est associé à un seul match.

3.2.7 Association Tournoi-Catégorie

- **Un tournoi appartient à une catégorie** : Un tournoi appartient à une seule catégorie (par exemple, "Homme" ou "Femme").
- **Une catégorie regroupe des tournois** : Une catégorie inclut 0 ou plusieurs tournois.

3.2.8 Association Catégorie-Gain

- **Une catégorie définit des gains** : Une catégorie peut avoir un ou plusieurs gains.
- **Un gain est associé à une catégorie** : Un gain appartient à une seule catégorie.

3.3 Structuration des tables

Les tables et leurs relations sont définies comme suit :

```sql
CREATE TABLE Categorie (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom VARCHAR(50) NOT NULL
);

CREATE TABLE Joueur (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    nationalite VARCHAR(50) NOT NULL,
    sponsor VARCHAR(100),
    classement_atp INTEGER NOT NULL
);

CREATE TABLE Tournoi (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom VARCHAR(100) NOT NULL,
    date_debut DATE NOT NULL,
    date_fin DATE,
    categorie_id INTEGER NOT NULL,
    FOREIGN KEY (categorie_id) REFERENCES Categorie(id)
);

CREATE TABLE Matchs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    tournoi_id INTEGER NOT NULL,
    joueur1_id INTEGER NOT NULL,
    joueur2_id INTEGER NOT NULL,
    tour VARCHAR(50) NOT NULL,
    niveau_tour INTEGER NOT NULL,
    gagnant_id INTEGER,
    nombre_sets INTEGER NOT NULL,
    FOREIGN KEY (tournoi_id) REFERENCES Tournoi(id),
    FOREIGN KEY (joueur1_id) REFERENCES Joueur(id),
    FOREIGN KEY (joueur2_id) REFERENCES Joueur(id),
    FOREIGN KEY (gagnant_id) REFERENCES Joueur(id)
);
```

```sql

CREATE TABLE Score (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    match_id INTEGER NOT NULL,
    set_numero INTEGER NOT NULL,
    jeux_joueur1 INTEGER NOT NULL,
    jeux_joueur2 INTEGER NOT NULL,
    FOREIGN KEY (match_id) REFERENCES Matchs(id)
);

CREATE TABLE Gain (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    categorie_id INTEGER NOT NULL,
    niveau_tour INTEGER NOT NULL,
    montant DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (categorie_id) REFERENCES Categorie(id)
);
```

3.4 Création de la base

Voici un exemple de création des tables pour la base de données.

```sql
-- Cr ation des cat gories
INSERT INTO Categorie (nom) VALUES ('Homme'),
   ('Femme'),('Juniors');

-- Cr ation des joueurs
INSERT INTO Joueur (nom, prenom, nationalite, sponsor,
   classement_atp) VALUES
('Haddad', 'Khalil', 'Liban', 'Cedars Tech', 150),
('Bensalah', 'Amar', 'Alg rie', 'Sahara Sport', 145);

-- Cr ation du tournoi
INSERT INTO Tournoi (nom, date_debut, date_fin, categorie_id)
   VALUES
('Grand Slam 2024', '2024-06-01', '2024-06-14', 1);

-- Cr ation des matchs
INSERT INTO Matchs (tournoi_id, joueur1_id, joueur2_id, tour,
   niveau_tour, gagnant_id, nombre_sets) VALUES
(1, 1, 2, '1er tour', 1, 1, 3);
```

3.5 Insertion des informations

Les données pour les scores et les gains sont insérées comme suit :

```sql
-- Scores
INSERT INTO Score (match_id, set_numero, jeux_joueur1,
   jeux_joueur2) VALUES
(1, 1, 6, 4), (1, 2, 3, 6), (1, 3, 7, 5);

-- Gains
INSERT INTO Gain (categorie_id, niveau_tour, montant) VALUES
```

```
(1, 1, 5000.00), (1, 2, 7500.00), (1, 3, 10000.00);
```

3.6 État de la base de données

Les tables sont peuplées avec un nombre minimal d'informations pour valider les cas d'usage.

4 Cas d'Usages

Dans cette section, nous définissons plusieurs cas d'usage pour tester et valider la base de données du projet. Ces cas d'usage incluent des recherches d'informations et des traitements sur les données.

4.1 Recherche d'Informations

4.1.1 Exemple 1 : Liste des joueurs d'un tournoi donné

Cas d'usage : Récupérer les informations des joueurs participant au tournoi "Grand Slam 2024".

Explication : 1. Identifier le tournoi ayant le nom "Grand Slam 2024". 2. Lier ce tournoi aux_matchs qu'il contient. 3. Extraire les joueurs impliqués dans ces matchs.

Requête SQL :

```
SELECT DISTINCT j.id, j.nom, j.prenom, j.nationalite
FROM Joueur j
INNER JOIN Matchs m ON j.id = m.joueur1_id OR j.id = m.joueur2_id
INNER JOIN Tournoi t ON m.tournoi_id = t.id
WHERE t.nom = "Grand Slam 2024";
```

—

4.1.2 Exemple 2 : Trouver le vainqueur d'un match spécifique

Cas d'usage : Identifier le joueur gagnant du match ayant l'identifiant `match_id = 3`.

Explication : 1. Filtrer le match ayant l'identifiant 3. 2. Lier ce match au joueur qui a gagné (attribut `gagnant id`).

Requête SQL :

```
SELECT j.nom, j.prenom
FROM Joueur j
INNER JOIN Matchs m ON j.id = m.gagnant_id
WHERE m.id = 3;
```

—

4.1.3 Exemple 3 : Scores détaillés d'un match donné

Cas d'usage : Afficher les scores par set pour le match ayant l'identifiant `match_id = 2`.

Explication : 1. Sélectionner les scores correspondant au match ayant `match id` 2. 2. Renvoyer les informations des sets joués.

Requête SQL :

```
SELECT s.set_numero, s.jeux_joueur1, s.jeux_joueur2
FROM Score s
WHERE s.match_id = 2;
```

4.1.4 Exemple 4 : Nombre total de sets joués dans un tournoi

Cas d'usage : Calculer le nombre total de sets joués dans le tournoi "Grand Slam 2024".

Explication : 1. Filtrer les matchs du tournoi "Grand Slam 2024". 2. Faire la somme du nombre de sets joués dans ces matchs.

Requête SQL :

```
SELECT SUM(m.nombre_sets) AS total_sets
FROM Matchs m
INNER JOIN Tournoi t ON m.tournoi_id = t.id
WHERE t.nom = "Grand Slam 2024";
```

4.1.5 Exemple 5 : Gains totaux par joueur

Cas d'usage : Calculer les gains totaux pour chaque joueur en fonction des matchs qu'il a remportés.

Explication : 1. Associer les matchs gagnés aux niveaux de tour correspondants pour obtenir les montants gagnés. 2. Grouper par joueur et calculer la somme des montants.

Requête SQL :

```
SELECT j.nom, j.prenom, SUM(g.montant) AS total_gain
FROM Joueur j
INNER JOIN Matchs m ON j.id = m.gagnant_id
INNER JOIN Gain g ON m.niveau_tour = g.niveau_tour
GROUP BY j.id;
```

4.1.6 Exemple 6 : Classement des joueurs par leur nationalité

Cas d'usage : Classer les joueurs par nationalité en affichant leur nombre et leur nationalité.

Explication : 1. Grouper les joueurs par nationalité. 2. Compter le nombre de joueurs pour chaque nationalité.

Requête SQL :

```
SELECT j.nationalite, COUNT(j.id) AS nb_joueurs
FROM Joueur j
GROUP BY j.nationalite;
```

4.1.7 Exemple 7 : Liste des joueurs sélectionnés pour la finale

Cas d'usage : Obtenir les informations des joueurs ayant participé à la finale d'un tournoi donné.

Explication : 1. Identifier le match de la finale. 2. Extraire les informations des joueurs y ayant participé.

Requête SQL :

```
SELECT DISTINCT j.nom, j.prenom, j.nationalite
FROM Joueur j
INNER JOIN Matchs m ON j.id = m.joueur1_id OR j.id = m.joueur2_id
WHERE m.tour = "Finale";
```

5 Mises à jour par des utilisateurs et par programmation

5.1 De la table au tableur

Pour permettre à un utilisateur de manipuler les données contenues dans la base de données, nous avons mis en place un programme Python permettant d'exporter une table au format CSV. Ce fichier peut ensuite être ouvert et modifié dans des outils classiques comme LibreOffice ou Microsoft Excel.

Objectif : Exporter une table au format CSV pour manipulation.

Algorithme :

1. Ouvrir une connexion à la base de données `Tennis.db`.
2. Sélectionner les données de la table souhaitée.
3. Écrire les données dans un fichier CSV.

Code Python :

```
import sqlite3
import csv

connection = sqlite3.connect("Tennis.db")
cursor = connection.cursor()

# Exportation des données
query = "SELECT * FROM table_name"
cursor.execute(query)
rows = cursor.fetchall()

# Création du fichier CSV
with open("table.csv", "w", newline="") as file:
    writer = csv.writer(file)
    # Écrire les en-têtes
    writer.writerow([description[0] for description in cursor.description])
    # Écrire les données
    writer.writerows(rows)

connection.close()
```

Ce programme génère un fichier `table.csv` contenant toutes les données de la table `table_name`. L'utilisateur peut modifier ce fichier dans un tableur.

—

5.2 Du tableur à la table

Une fois que les modifications nécessaires ont été apportées au fichier CSV, les données peuvent être réintégrées dans la base de données. Nous avons développé un programme Python qui permet de lire un fichier CSV et de mettre à jour ou insérer les données dans la table correspondante de la base `Tennis.db`.

Objectif : Importer un fichier CSV dans une table de la base de données.

Algorithme :

1. Ouvrir une connexion à la base de données `Tennis.db`.
2. Lire les données depuis le fichier CSV.
3. Insérer ou mettre à jour les données dans la table.

Code Python :

```
import sqlite3
import csv

connection = sqlite3.connect("Tennis.db")
cursor = connection.cursor()
```

```
# Lecture des données depuis le fichier CSV
with open("table.csv", "r") as file:
    reader = csv.reader(file)
    headers = next(reader)  # Lire les en-têtes
    placeholders = ", ".join(["?" for _ in headers])
    query = f"INSERT OR REPLACE INTO table_name ({', '.join(headers)})

    VALUES ({placeholders})"

    # Insérer les données
    for row in reader:
        cursor.execute(query, row)

connection.commit()
connection.close()
```

Ce programme suppose que les en-têtes du fichier CSV correspondent exactement aux noms des colonnes dans la table cible. Si une ligne existe déjà (selon la clé primaire ou unique), elle est mise à jour ; sinon, une nouvelle ligne est insérée.

5.3 Calcul des scores

Le calcul des scores des matchs est basé sur les sets remportés par les joueurs. Un programme Python calcule les scores automatiquement en se basant sur les données des sets enregistrées dans la table `Score`.

Objectif : Calculer les scores des joueurs et mettre à jour les résultats dans la table `Matchs`.

Algorithme :

1. Pour chaque match, récupérer les sets associés.
2. Calculer les jeux gagnés pour chaque joueur.
3. Mettre à jour les scores des joueurs dans la table `Matchs`.

Code Python :

```
import sqlite3

# Connexion à la base de données
connection = sqlite3.connect("Tennis.db")
cursor = connection.cursor()

# Calcul des scores
query = "SELECT match_id FROM Matchs"
cursor.execute(query)
matches = cursor.fetchall()

for match in matches:
    match_id = match[0]
```

```
    # Récupération des scores des sets
    query = "SELECT jeux_joueur1, jeux_joueur2 FROM Score WHERE match_id = ?"
    cursor.execute(query, (match_id,))
    scores = cursor.fetchall()

    # Calcul des scores totaux
    joueur1_score = sum(row[0] for row in scores)
    joueur2_score = sum(row[1] for row in scores)

    # Mise à jour des scores dans la table Matchs
    update_query = "UPDATE Matchs SET joueur1_score = ?, joueur2_score = ? WHERE id =
    cursor.execute(update_query, (joueur1_score, joueur2_score, match_id))

connection.commit()
connection.close()
```

—

5.4 Applications pratiques

Ces programmes permettent de répondre à plusieurs besoins pratiques :

- Extraction des données pour une analyse ou une modification dans un tableur.
- Mise à jour automatisée des scores après l'insertion des sets.
- Intégration des modifications dans la base de données

6 Conclusion

Dans ce projet, nous avons réussi à modéliser un tournoi de tennis en répondant aux besoins définis dans le cahier des charges. À partir d'une analyse détaillée des entités et des relations entre elles, nous avons conçu une base de données relationnelle structurée et efficace.

Les cas d'usage définis nous ont permis de valider notre modèle de données en manipulant et en interrogeant les informations via des requêtes SQL. Nous avons également mis en place des outils pratiques pour les utilisateurs non techniques, leur permettant de manipuler les données à l'aide de fichiers CSV et de tableurs. Ces outils garantissent une grande flexibilité et une accessibilité accrue des informations.

En complément, des programmes d'application ont été développés pour automatiser les calculs des scores des joueurs et la mise à jour des résultats des matchs. Ces fonctionnalités renforcent la fiabilité et l'efficacité du système en minimisant les erreurs manuelles et en optimisant les opérations répétitives.

Pour aller plus loin, plusieurs pistes d'amélioration peuvent être envisagées :

- Intégrer des statistiques avancées, telles que le pourcentage de victoires des joueurs ou le temps total des matchs.
- Ajouter une gestion des événements spécifiques, comme les abandons, les blessures ou les pénalités.

- Mettre en place une interface utilisateur graphique (IHM) pour rendre le système plus intuitif.
- Automatiser l'importation des résultats en temps réel à partir de sources externes.

Ce projet a permis de combiner les concepts fondamentaux des bases de données relationnelles avec des outils de programmation pratiques. Il constitue une base solide pour la gestion de tournois de tennis et peut être facilement étendu pour répondre à de nouvelles exigences ou à des sports similaires.