Babyfoot Connecté – Schéma SQL & Documentation

Présentation générale

Cette base de données a été conçue pour moderniser et enrichir l'expérience du **babyfoot à Ynov Toulouse**, dans le cadre d'un projet **IA & Data** visant à rendre le babyfoot :

- **Connecté**, grâce à des données en temps réel (scores, durée, ambiance, performances),
- **Analytique**, avec suivi des joueurs et statistiques détaillées,
- **Ludique**, via un système de **paris internes** simulant une expérience de betting friendly.

Le modèle repose sur deux grands modules :

1. ** Module Babyfoot Connecté** — gestion des matchs, joueurs, tables et télémétrie.
2. ** Module Paris Sportifs** — gestion des parieurs, types de paris et règlements automatiques.

---

1. Module Babyfoot Connecté

Table `players`

Contient les informations sur chaque joueur.

Colonne	Type	Description
`player_id`	`VARCHAR(16)`	Identifiant unique
`name`	`TEXT`	Nom complet
`age`	`INT`	Âge
`notes`	`TEXT`	Observations (niveau, main dominante, etc.)

Pourquoi ?
Nous avons isolé les joueurs dans une table dédiée pour permettre :

- la réutilisation des profils dans plusieurs matchs,
- la possibilité d'ajouter ultérieurement des statistiques agrégées (victoires, ratio, ELO...).

Table `tables_foos`

Liste les tables de babyfoot présentes sur le campus.

Colonne	Type	Description
`table_id`	`VARCHAR(8)`	Identifiant unique de la table
`location`	`TEXT`	Lieu (ex : "Souk Ynov 1")
`condition`	`TEXT`	État de la table
`ball_type`	`TEXT`	Type de balle utilisée

Pourquoi ?
Chaque table est unique (lieu, usure, modèle).
Cela permet :

- d'étudier l'utilisation de chaque table,
- de planifier la maintenance,
- ou de croiser la performance selon le matériel utilisé.

Table `games`

Représente un match joué.

Colonne	Type	Description
`game_id`	`VARCHAR(16)`	Identifiant du match
`game_datetime`	`TIMESTAMP`	Date et heure
`location`	`TEXT`	Lieu
`table_id`	`VARCHAR(8)`	Référence à `tables_foos`
`score_red` / `score_blue`	`INT`	Scores finaux
`winner`	`TEXT`	“Red” ou “Blue”
`duration_s`	`INT`	Durée en secondes
`attendance`	`INT`	Nombre de spectateurs
`season`	`TEXT`	Saison ou tournoi
`recorded_by`	`TEXT`	Auteur de la saisie ou du capteur

Pourquoi ?
Le cœur de la BDD. Cette table centralise tous les matchs.
Elle relie :

- la table physique (`table_id`),
- les joueurs (via `game_players`),
- les données techniques (`telemetry`),
- et les paris (`bets`).

Table `game_players`

Associe un joueur à un match, avec ses stats individuelles.

Colonne	Type	Description
`game_id`	`VARCHAR(16)`	Référence à `games`
`player_id`	`VARCHAR(16)`	Référence à `players`
`team`	`TEXT`	“Red” ou “Blue”
`role`	`TEXT`	Poste (attaque/défense)
`goals`	`INT`	Buts marqués
`own_goals`	`INT`	CSC
`assists`	`INT`	Passes décisives
`saves`	`INT`	Arrêts
`possession_s`	`INT`	Temps de possession
`is_substitute`	`BOOLEAN`	Remplaçant ou non

Pourquoi ?
Cette table fait le lien **N-N** entre joueurs et matchs, et stocke les **statistiques détaillées**.
Cela permet des analyses comme :

- “Qui marque le plus de buts par match ?”
- “Quel joueur défend le mieux ?”

Table `telemetry`

Stocke les données techniques et contextuelles.

Colonne	Type	Description
`game_id`	`VARCHAR(16)`	Référence au match
`ping_ms`	`INT`	Latence mesurée
`music_playing`	`TEXT`	Musique jouée pendant le match
`referee`	`TEXT`	Arbitre (bot ou humain)
`created_dt`	`TIMESTAMP`	Date d'enregistrement

Pourquoi une colonne de latence (`ping_ms`) ?
Même si le babyfoot est physique, il peut être **connecté à une application web** :

- pour afficher les scores en direct,
- pour envoyer des événements (buts, pauses, stats) à un serveur,
- ou pour synchroniser des données vers un dashboard.

La **latence** permet donc de **mesurer la qualité de la connexion** entre la table connectée (ou le Raspberry associé) et le serveur central — utile pour la fiabilité du système et le diagnostic réseau.

---

2. Module Paris Sportifs

Table `bettors`

Liste les utilisateurs pariant sur les matchs.

Colonne	Type	Description
`bettor_id`	`SERIAL`	Identifiant
`username`	`TEXT`	Identifiant unique
`display_name`	`TEXT`	Nom affiché
`balance_cents`	`BIGINT`	Solde virtuel (en centimes)
`created_at`	`TIMESTAMP`	Date d'inscription

Pourquoi ?
Séparer les parieurs des joueurs permet :

- d'avoir des utilisateurs non-joueurs (spectateurs, étudiants),
- de simuler un système de points interne,
- de garder une cohérence entre profils utilisateurs et joueurs physiques.

Table `bet_types`

Catalogue des types de paris disponibles.

Colonne	Type	Description
`bet_type_id`	`VARCHAR(32)`	Identifiant
`description`	`TEXT`	Description

Pourquoi ?
Cette table permet d'ajouter facilement de nouveaux types de paris sans modifier la structure (ex : nombre de passes, ratio de buts, durée moyenne...).

Table `bets`

Historique des paris placés.

Colonne	Type	Description
`bet_id`	`VARCHAR(32)`	Identifiant
`bettor_id`	`INT`	Référence à `bettors`
`game_id`	`VARCHAR(16)`	Référence à `games`
`bet_type_id`	`VARCHAR(32)`	Type de pari
`stake_cents`	`BIGINT`	Mise (en centimes)
`odds_num` / `odds_den`	`BIGINT`	Cote fractionnelle
`placed_at`	`TIMESTAMP`	Date du pari
`settled`	`BOOLEAN`	Pari réglé ou non
`won`	`BOOLEAN`	Gagné ou perdu
`payout_cents`	`BIGINT`	Gain versé

Pourquoi ?
Séparer la table des paris des sélections permet d'avoir :

- des paris simples (un seul choix),
- des combinés (plusieurs sélections).

Table `bet_selections`

Permet plusieurs sélections dans un même pari.

Colonne	Type	Description
`selection_id`	`SERIAL`	Identifiant
`bet_id`	`VARCHAR(32)`	Référence à `bets`
`selection_key`	`TEXT`	Détail (ex : `winner:Red`)
`selection_value`	`TEXT`	Valeur complémentaire
`odds_num` / `odds_den`	`BIGINT`	Cote spécifique

Fonction `settle_moneyline_bets(p_game_id)`

Fonction PostgreSQL permettant de régler automatiquement les paris de type **moneyline** (vainqueur du match).

Principe :

1. Récupère le vainqueur réel dans `games.winner`.
2. Parcourt tous les paris non réglés pour ce match.
3. Vérifie s'ils contiennent une sélection `winner:Red` ou `winner:Blue`.
4. Calcule le gain selon la cote fractionnelle : payout = stake_cents * (odds_num / odds_den)
5. Met à jour le solde du parieur et marque le pari comme réglé.

Choix de conception

Décision	Justification
Base relationnelle PostgreSQL	Meilleure cohérence, relations claires et typage fort pour le reporting et les fonctions PL/pgSQL.
Séparation des modules (babyfoot / paris)	Permet une évolution indépendante : on peut déployer le module “paris” plus tard ou le désactiver sans casser le cœur sportif.
Utilisation de `VARCHAR` courts pour les IDs	Légers à transférer, compatibles avec des identifiants externes (API, IoT, applis mobiles).
Stockage des montants en centimes (`BIGINT`)	Évite les erreurs d’arrondi liées aux flottants dans les systèmes de mise/pari.
Table `telemetry`	Données techniques (ping, musique, arbitre) utiles à la fois pour le diagnostic et pour la création d’analyses d’ambiance (par exemple : “les matchs avec musique durent plus longtemps”).
Index ciblés sur les dates, vainqueurs et tables	Optimisation des statistiques (classements, historique, filtrage).
Fonction `settle_moneyline_bets`	Automatisation du traitement des paris gagnants/perdants sans intervention humaine.

En résumé

Cette base de données permet de :

- Suivre **toutes les activités liées au babyfoot connecté** (matchs, stats, ambiance)
- Gérer **les joueurs** et leurs performances
- Offrir une dimension **ludique et sociale** avec des paris internes simulés
- Préparer des **analyses IA/Data** pour l’amélioration continue des performances et de l’expérience

> Ce modèle est pensé pour évoluer : ajout de tournois, IA de prédiction, badges, leaderboard, ou intégration à une app mobile.