Enseirb-Matmeca

Informatique 2^e année

Fédération Sportive de Basket-ball

Date : 12 Décembre 2014

Table des matières

Introduction

L'objectif de ce projet est de réaliser une base de données gérant les rencontres de Basket-Ball entre les différents clubs d'une fédération. Un club est constitué de plusieurs entités telles que :

— un bureau ;
— des équipes ;
— des joueurs ;
— des entraineurs ;
— des catégories.

Les clubs de la fédération réalisent des rencontres à une certaine date. On veut pouvoir stocker les statistiques des joueurs participant à ces rencontres. Notre base de données doit donc stocker toutes ces informations. Elle doit également pouvoir être consultée, mise à jour ou détruite.

Afin de créer la base de données réalisant cette fédération, nous avons élaboré plusieurs diagrammes Entité-Association.

1 Modélisation des données

Dans cette partie, nous présentons les diagrammes que nous avons réalisés lors de l'élaboration de notre base. Nous commençons par notre diagramme entité-association initial, puis nous exposons le modèle entité-association retenu.

1.1 Diagramme Entité-Association

1.1.1 Modèle initial

Nous avons commencé par vouloir regrouper un maximum d'informations dans différentes tables. Vous trouverez en figure 1 le premier modèle que nous avions élaboré.

Les problèmes principaux étaient les nombreuses contraintes d'intégrité fonctionnelle entre les différentes entités du modèle. Par exemple, un joueur ne pouvait pas être un entraîneur et inversement. D'autre part, nous stockions le nombre de matchs gagnés, perdus et nuls dans l'entité équipe. Ces informations étaient redondantes, car il est possible de les retrouver en utilisant les attributs «score» de l'entité rencontre. Sur ce premier modèle, nous avions également une unique équipe reliée à l'entité rencontre. Cela pose problème, car deux équipes exactement doivent participer à une rencontre. Il aurait fallu ajouter une clef étrangère aux équipes pour retrouver une rencontre. Mais dans ce cas, il fallait aussi vérifier que deux équipes exactement possédaient une même clef étrangère.

Dans ce diagramme, l'entité catégorie était peu utile, en effet chaque équipé ayant un identifiant unique, il est possible de stocker le nom de catégorie dans l'entité équipe. Il n'y pas de problème pour un club à avoir plusieurs équipes dans une même catégorie.

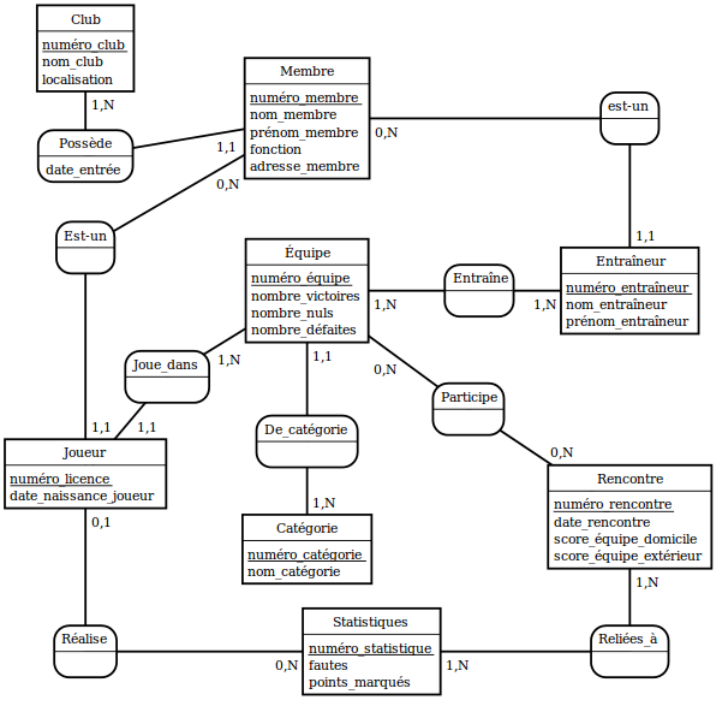

FIGURE 1 – Modèle Entité-Association initial

En raison de ces contraintes, nous avons cherché à obtenir un modèle plus ergonomique.

1.1.2 Modèle retenu

Le modèle retenu est présenté en figure 2. Les principaux changements ont été de déplacer certains attributs et de séparer les membres d'un club qui sont membres du bureau et ceux qui sont entraineurs et/ou joueurs. (Les membres du bureau peuvent également être joueurs et/ou entraîneurs).

Les **hypothèses** que nous avons faites sont les suivantes :

— chaque responsable d'un club exerce une et une seule fonction. Par exemple, le président d'un club ne peut pas être trésorier ce club.
— A priori, un entraîneur ne peut pas être un joueur. On peut cependant avoir un joueur-entraîneur dans un club. Le défaut est que son nom, son prénom et sa date d'entrée sont présents en double dans la base. Cette situation étant très rare, nous n'avons pas souhaité faire de modification sur ce point.
— Une rencontre se fait entre deux équipes qui sont de même catégorie.
— Les joueurs d'une équipe peuvent ne pas participer à une rencontre, ils ne possèdent alors pas de statistiques.
— Un entraineur peut entraîner une ou plusieurs équipes mais dans un seul club.

Les autres choix que nous avons fait concernent les attributs de certaines entités.

En ce qui concerne les *rencontres*, nous avons décidé de les lier à deux équipes. Nous avons choisi de stocker le score de chacune des équipes dans cette entité, car

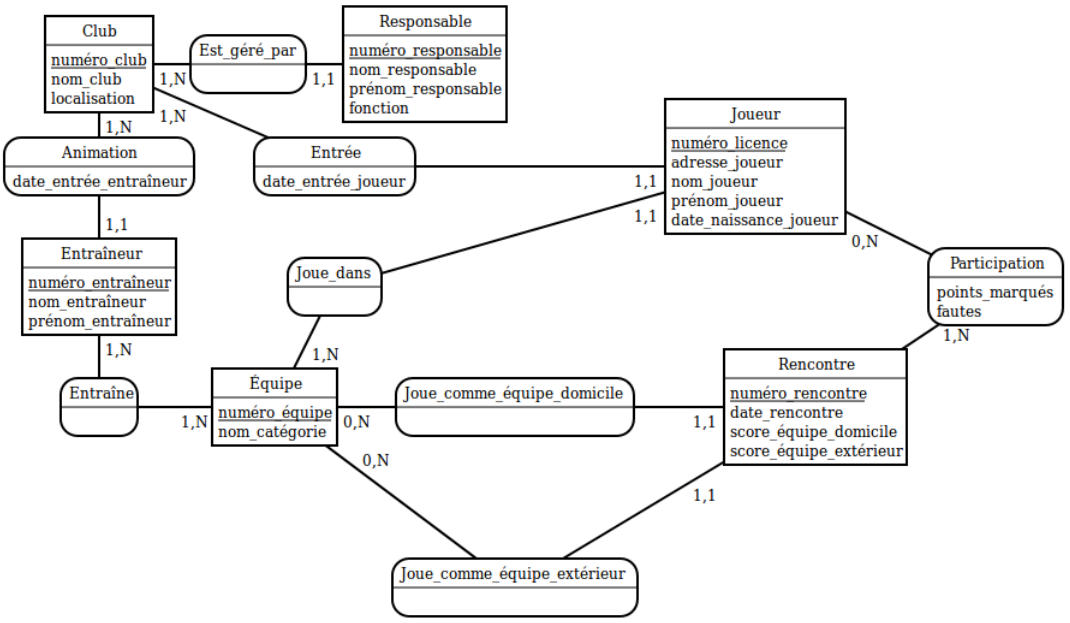

FIGURE 2 – Modèle Entité-Association retenu

cela permet de simplifier certaines requêtes de statistiques. Par exemple, sans ces attributs pour vérifier la victoire d'une équipe, il faudrait additionner les points marqués par chaque joueur de l'équipe lors de la rencontre donnée. Pour compter plusieurs victoires, cela devient rapidement compliqué. Nous avons aussi ajouté un attribut journée dans les rencontres. Une journée (chiffre) définit une plage de date où un ensemble de rencontres est joué dans le championnat par plusieurs équipes. Cet attribut nous semble pertinent pour réaliser des requêtes sur plusieurs rencontres. Par exemple en cas de rencontre reportée, on peut tout de même prendre en compte dans le classement le résultat des équipes pour la journée de cette rencontre.

Pour les *catégories*, nous avons décidé de les stocker dans l'entité équipe. En effet, chaque équipe ayant un identifiant unique, il est possible de stocker la catégorie dans cette équipe. Plusieurs équipes dans un même club peuvent être dans la même catégorie sans problème.

Les tables *animation* et *entree* ont été créés pour faciliter les requêtes de mise à jour qui concerne les entraîneurs et joueurs. Si un entraîneur (respectivement un joueur) change de club alors seule la table *animation* (respectivement *entree*) et l'attribut numéro_entraineur (respectivement numéro_licence) sont à changer.

2 Diagramme Relationnel

Un des objectifs du projet était d'obtenir un modèle relationnel en troisième forme normale. Il a donc fallu vérifier plusieurs choses :

— La clef de chaque entité doit être unique ;

— chaque attribut d'une entité doit dépendre de toute la clef de l'entité et non d'une partie de cette clef seulement ;
— chaque attribut d'une entité doit pouvoir être déterminé uniquement par la clef de l'entité. Si un autre attribut que la clef permet de déterminer un autre attribut de l'entité alors le modèle n'est pas en troisième forme normale.

Le modèle relationnel correspondant au schéma entité-association retenu et respectant les contraintes décrites ci-dessus est présenté en figure 3.

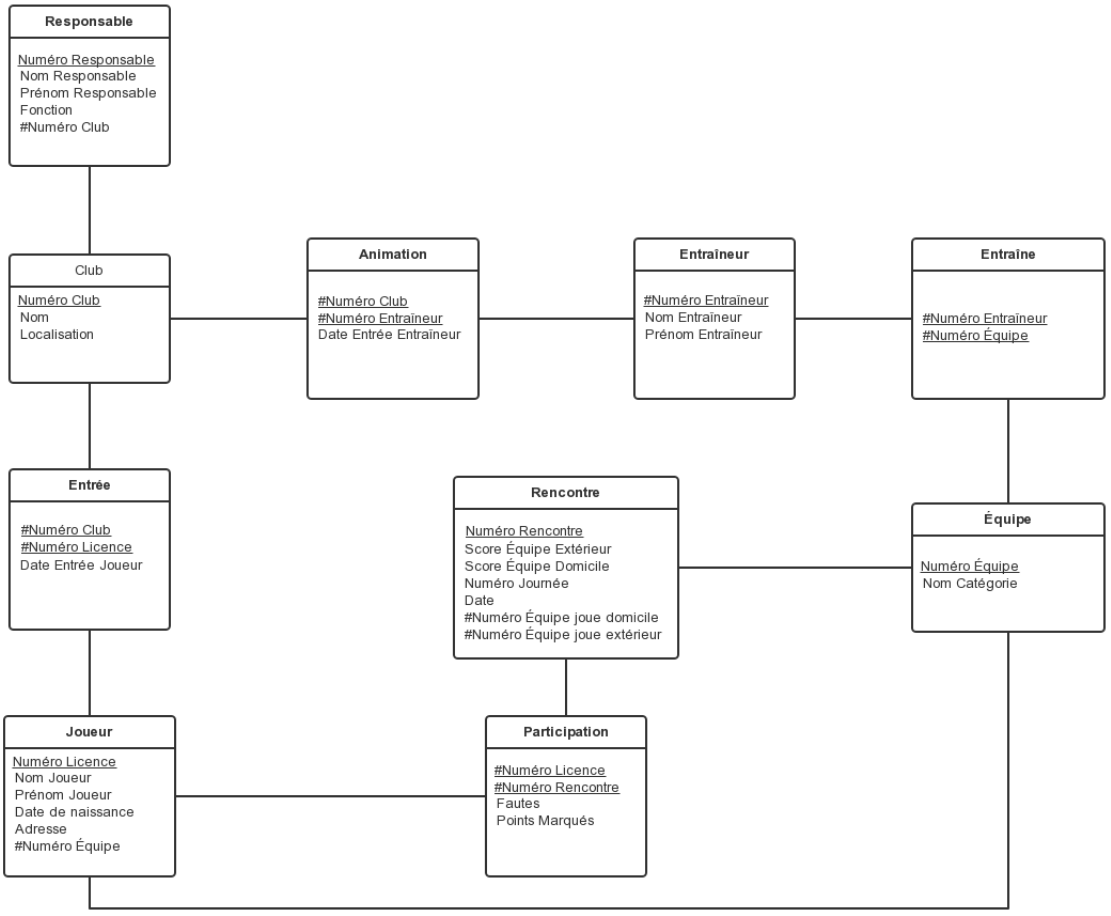

Figure 3 – Modèle Relationnel

CLUB	(numéro_club, *nom_club*, *localisation*)
RESPONSABLE	(numéro_responsable, *nom_responsable*, *prénom_responsable*, *fonction*, #*numéro_club*)
ENTRAÎNEUR	(numéro_entraîneur, *nom_entraîneur*, *prénom_entraîneur*, #*numéro_club*, *date_entrée_entraîneur*)
ANIMATION	(#*numéro_club*, #*numéro_entraîneur*, *date_entrée_entraîneur*)
ENTRAÎNE	(#*numéro_équipe*, #*numéro_entraîneur*)
ÉQUIPE	(numéro_équipe, *nom_catégorie*)
JOUEUR	(numéro_licence, *adresse_joueur*, *nom_joueur*, *prénom_joueur*, *date_naissance_joueur*, #*numéro_équipe*, #*numéro_club*, *date_entrée_joueur*)
ENTRÉE	(#*numéro_club*, #*numéro_joueur*, *date_entrée_joueur*)
PARTICIPATION	(#*numéro_licence*, #*numéro_rencontre*, *points_marqués*, *fautes*)
RENCONTRE	(numéro_rencontre, *date_rencontre*, *score_équipe_domicile*, *score_équipe_extérieur*, #*numéro_équipe.1*, #*numéro_équipe.2*)

3 Implémentation

Nous avons utilisé une base de données MySQL. Les requêtes sont disponibles dans des fichiers sql. Dans cette partie, nous présentons uniquement certaines des requêtes que nous avons réalisées.

3.1 Requêtes de création/suppression

Les requêtes de création des tables sont classiques. Nous allons détailler le fonctionnement de la création d'une table sur la table CLUB.

Nous vérifions d'abord qu'une table CLUB n'est pas déjà existante. Si une telle table existe, on la supprime :

```
drop table if exists CLUB cascade;
```

On crée ensuite la table correspondant à notre modèle relationnel, on spécifie quelle est la clef primaire, on ajoute la condition qu'elle ne peut pas être nulle et qu'elle s'incrémentera automatiquement en cas d'ajout de données à cette table :

```
CREATE TABLE club
(
numero_club   INT(11) NOT NULL AUTO_INCREMENT,
nom_club CHAR(20),
localisation CHAR(20),
    CONSTRAINT PK_ACTEUR PRIMARY KEY (numero_club)
);
```

Listing 1 – Initialisation Club

En ce qui concerne l'ajout de données, il fallait faire attention à l'ordre dans lequel les entités étaient créées en raison de la création de clefs étrangères dans certaines entités. Nous avons donc ajouté en premier les entités qui ne possédaient pas de clefs étrangères, puis celles qui possédaient une clef étrangère vers une des classes créées. Voici l'exemple d'ajout des clubs :

```
INSERT INTO club VALUES ( 1 , "Lakers" , "Los␣Angeles" ) ;
INSERT INTO club VALUES ( 2 , "Heat" , "Miami" ) ;
INSERT INTO club VALUES ( 3 , "Bulls" , "Chicago" ) ;
INSERT INTO club VALUES ( 4 , "Spurs" , "San␣Antonio" ) ;
INSERT INTO club VALUES ( 5 , "Celtics" , "Boston" ) ;
COMMIT ;
```

Listing 2 – Ajout des equipes

3.2 Requêtes de mise à jour

Les instructions pour réaliser les requêtes de mise à jour sont très similaires à celles d'ajout, il suffit de spécifier les champs que l'on veut modifier et remplacer le mot clef *insert* par le mot clef *update*.

Voici un exemple de mise à jour sur la table club :

```
UPDATE club
SET club.localisation = "New␣York"
WHERE club.numero_club = 1;
```

Listing 3 – Modification de la localisation d'un club

Il suffit alors de vérifier l'intégrité des données à l'aide de déclencheurs.

3.3 Requêtes de consultation

3.3.1 Informations générales

Dans cette partie, nous allons présenter certaines des requêtes de consultation qui fournissent des informations sur différents éléments d'un club ou des informations sur tous les clubs de la fédération.

Les requêtes les plus simples sont celles qui permettent d'obtenir des informations sur l'entité club. Il suffit de saisir simplement : *SELECT * FROM* club
On peut alors affiner la demande en spécifiant un numéro de club ou un nom de club ou une localisation.

On peut réaliser des requêtes similaires sur les entités rencontre, responsable, joueur, entraîneur pour obtenir des informations comme les noms et prénom des

membres d'un club.
SELECT * FROM joueur fournit ainsi la listes des noms, prénoms, addresse, dates de naissance de tous les joueurs de la fédération. On peut affiner ces requêtes en spécifiant certains critères comme le numéro de club, une date de naissance, une date d'entrée dans un club, *etc.*...

Voici quelques exemples :

```
SELECT joueur.*
FROM joueur, entree, club
WHERE club.numero_club = entree.numero_club
AND entree.numero_licence = joueur.numero_licence
```

Listing 4 – Liste des joueurs d'un club

```
SELECT joueur.*
FROM joueur, entree, club
WHERE club.numero_club = entree.numero_club
AND entree.numero_licence = joueur.numero_licence
AND entree.date_entree_joueur = "date_voulue"
```

Listing 5 – Liste des joueurs d'un club inscrits avant la «date voulue»

3.3.2 Statistiques

Les requêtes fournissant les statistiques ont été les plus compliquées à réaliser. Elles font généralement intervenir plusieurs tables intermédiaires. La difficulté principale était de relier la table rencontre avec les autres entités dont nous voulions extraire des informations. Parfois plus de quatre tables de notre base de données sont utilisées pour constituer la requête de statistique désiré. On les agence grâce à l'imbrication consécutive de select from(...)
La requête la plus globale est celle du classement des équipes intégrant le goalaverage de chacune d'entre-elles, pouvant les départager en cas d'égalité au nombre de points. Le système retenu est le suivant : 3 points en cas de victoire, 1 point en cas de match nul, 0 points en cas de défaite.

Pour des raisons de place, nous n'intégrerons pas dans ce rapport la requête de classement général des équipes d'une catégorie. Les exemples 1 et 2 sont des requêtes "passerelles" ayant permis de construire ce classement.

— *Exemple 1* : la requête permettant de connaitre le nombre de victoire à domicile de chaque équipe pour la catégorie senior, depuis le début de la saison. La catégorie est modifiable par l"utilisateur depuis l"interface de notre projet.

```
SELECT vic_dom.numero_equipe, nombre_victoire_domicile,
nom_categorie
FROM (SELECT numero_equipe, count(rencontre.numero_rencontre)
as nombre_victoire_domicile
    FROM equipe
```

```
    LEFT JOIN rencontre
    ON rencontre.numero_equipe_joue_domicile =
        equipe.numero_equipe
    AND score_equipe_domicile > score_equipe_exterieur
    GROUP BY numero_equipe) AS vic_dom, equipe

WHERE equipe.nom_categorie = "senior"
AND vic_dom.numero_equipe = equipe.numero_equipe
ORDER BY nombre_victoire_domicile DESC
```

Listing 6 – Victoire à domicile

— *Exemple 2* : la requête permettant de connaitre le goalaverage de chaque équipe, soit la somme sur toute la saison de la différence entre le score de l'équipe et le score de l'équipe adverse à chaque rencontre.

```
SELECT rtr_l.numero_equipe,
IFNULL(diff_domicile,0)+IFNULL(diff_exterieur,0)
AS goalaverage_general
FROM (SELECT numero_equipe,
SUM(score_equipe_domicile-score_equipe_exterieur)
AS diff_domicile
    FROM equipe
    LEFT JOIN rencontre
    ON rencontre.numero_equipe_joue_domicile
        = equipe.numero_equipe
    GROUP BY numero_equipe) AS goal_l
LEFT JOIN   (SELECT numero_equipe,
SUM(score_equipe_exterieur-score_equipe_domicile)
AS diff_exterieur
    FROM equipe
    LEFT JOIN rencontre
    ON rencontre.numero_equipe_joue_exterieur
        = equipe.numero_equipe
    GROUP BY numero_equipe) goal_r
ON goal_l.numero_equipe = goal_r.numero_equipe
ORDER BY goalaverage_general DESC
```

Listing 7 – Goalaverage de chaque équipe

Cette requête se construit selon la structure suivante :

1. Création de la table de goalaverage exterieur
2. Création de la table de goalaverage domicile
3. selection pour chaque équipe de leur goalaverage général (somme des colonnes diff_exterieur et diff_domicile de ces deux tables)

— *Exemple 3* : en dernier exemple des requêtes de statistiques, la requête présentant les meilleurs joueurs d'une catégorie pour une journée, classés en fonction

de leur points marqués et de leurs nombres de fautes commises. La catégorie et la journée sont choisis par l'utilisateur, ici égales respectivement à "junior" et 1.

```
SELECT DISTINCT nom_joueur, prenom_joueur,
participation.numero_licence, points, fautes
      FROM participation, joueur, rencontre, equipe
             WHERE joueur.numero_licence
                      = participation.numero_licence
       AND rencontre.numero_rencontre
              = participation.numero_rencontre
       AND (joueur.numero_equipe = numero_equipe_joue_domicile
              or
           joueur.numero_equipe = numero_equipe_joue_exterieur)
       AND (equipe.numero_equipe = numero_equipe_joue_domicile
              or
            equipe.numero_equipe = numero_equipe_joue_exterieur)
       AND equipe.nom_categorie = "junior"
       AND numero_journee = 1
GROUP BY nom_joueur
ORDER BY points DESC, fautes ASC
```

Listing 8 – Goalaverage de chaque équipe

4 Interface

L'interface fait intervenir le language PHP. Nous avons pris un soin particulier à avoir une organisation structurée du code (suivant autant que possible un pattern MVC), respectieuse à un niveau correct des principes élémentaires d'une application web (par effectuer une redirection après avoir soumis un formulaire), en mettant en pratique certains principes enseignés dans le module Web/XML.

4.1 Installation

Pour l'installation, il faut vérifier en premier lieu que l'extension `pdo_mysql` est activée, et qu'une base de données MySQL est disponible.

Les identifiants de la base de données doivent être renseignés dans le fichier `app/config.php`.

Un bouton est alors disponible sur la page d'accueil pour créer les différentes tables ainsi que charger les fixtures.

Par ailleurs, une version mirroir est disponible à l'adresse .

4.2 Organisation du code

Le code est partagé entre trois répertoires principaux : le premier est `src` contenant les sources, le second est `app` contenant la configuration de l'application, et enfin

`web` contenant les contrôleurs (on se base sur le routage induit par l'arborescence des fichiers).

L'arborescence des fichiers sources est la suivante :

1. `models` contient les modèles des différentes entités d'une part, et les *repository* associés à chacune des entités permettant de regrouper certaines requêtes d'autre part,
2. `templates` contenant des fichiers .php équivalents à des templates, au sens où on ne fait que de l'affichage,
3. `modules` contenant quelques utilitaires, notamment pour partager la connexion à la base de données ainsi qu'une fonction permettant de faire le rendu d'un template.

4.3 Organisation du site

L'interface se concentre en premier sur le club et implémente depuis cette entité des accès aux différentes relations, en cascade. Des listes sont également disponibles pour chaque entité principale permettant d'accéder à des fonctions de suppression et d'édition lorsqu'elles sont implémentées. L'accent a été mis sur la simplicité de l'interface en évitant d'agréger beaucoup de données sur une seule page (en pratique uniquement les relations directes sont reflétées).

Concernant l'aspect visuel, nous avons utilisé le framework *Foundation* qui a la particularité de forcer une bonne structure du code HTML sans pour autant dénaturer sa nature première de transmission de données, et non de mise en forme de celle-ci. Ceci permet enfin d'avoir une interface propre sans perdre de temps à travailler sur le design.

5 Conclusion

Ce projet a été l'occasion de développer et d'approfondir les connaissances acquises en TD, à la fois en ce qui concerne les diagrammes entité-association que les diagrammes relationnels ou la maîtrise de MySQL. En premier lieu, il a fallu bien définir les limites du sujet afin d'obtenir un modèle cohérent de fédération de basket, chaque table devant respecter les 3 formes normales. Ensuite, la partie codage a été composée de l'ajout de la base suivant le fonctionnement du modèle créé, l'ajout de données suffisantes dans la base afin de satisfaire un certain nombre de requêtes, et l'ajout des requêtes demandées dans le sujet. Enfin, la partie interface réalisée en PHP consent à apporter un confort à l'utilisateur que ce soit pour l'ajout de données, la consultation des tables ou de requêtes de statistiques, notamment le classement général des équipes.