EPHEC 2019-2020

HENSMANS Olivier

baccalauréat en informatique de gestion

FifaManager

Projet pour le cours de SGBD de Monsieur Vincent Fievez

Table des matières

[1 Contexte 4](#_Toc37685105)

[1.1 Cadre 4](#_Toc37685106)

[1.2 Projet 4](#_Toc37685107)

[2 Analyse métier 5](#_Toc37685108)

[2.1 Description de la solution envisagée 5](#_Toc37685109)

[2.2 Intervenants 5](#_Toc37685110)

[3 Fonctions attendues 6](#_Toc37685111)

[3.1 Exigences fonctionnelles 6](#_Toc37685112)

[3.1.1 BackEnd 6](#_Toc37685113)

[3.1.2 MatchManagement 6](#_Toc37685114)

[3.2 Exigences non fonctionnelles 6](#_Toc37685115)

[4 Contraintes Business 7](#_Toc37685116)

[5 Analyse fonctionnelle 8](#_Toc37685117)

[5.1 Diagrammes de Use Cases 8](#_Toc37685118)

[5.1.1 BackEnd 8](#_Toc37685119)

[5.2 Championnats 13](#_Toc37685120)

[5.3 Équipes 13](#_Toc37685121)

[5.4 Joueurs 13](#_Toc37685122)

[5.5 Matchs 14](#_Toc37685123)

[5.6 Feuilles de match 14](#_Toc37685124)

[6 Entité Association 15](#_Toc37685125)

[7 Schéma Relationnel 16](#_Toc37685126)

[8 Tables SQL 17](#_Toc37685127)

[8.1 Avant-Propos 17](#_Toc37685128)

[8.2 Table Championnat 18](#_Toc37685129)

[8.2.1 Trigger : 18](#_Toc37685130)

[8.2.2 Procédures stockées : 18](#_Toc37685131)

[8.3 Table Intersaisons 19](#_Toc37685132)

[8.3.1 Trigger : 19](#_Toc37685133)

[8.3.2 Procédures stockées : 19](#_Toc37685134)

[8.4 Table Quarters 20](#_Toc37685135)

[8.4.1 Triggers : 20](#_Toc37685136)

[8.4.2 Procédures stockées : 20](#_Toc37685137)

[8.5 Table Equipes 21](#_Toc37685138)

[8.5.1 Triggers : 21](#_Toc37685139)

[8.5.2 Procédures stockées : 21](#_Toc37685140)

[8.6 Table EquipesParticipationHistory 22](#_Toc37685141)

[8.6.1 Trigger : 22](#_Toc37685142)

[8.6.2 Procédures stockées : 22](#_Toc37685143)

[8.7 Table Joueurs 23](#_Toc37685144)

[8.7.1 Procédures stockées 23](#_Toc37685145)

[8.8 Table Matchs 24](#_Toc37685146)

[8.8.1 Triggers : 25](#_Toc37685147)

[8.8.2 Procédures stockées : 25](#_Toc37685148)

[8.9 Table CartonsRougesHistory 26](#_Toc37685149)

[8.9.1 Triggers : 26](#_Toc37685150)

[8.9.2 Procédures stockées : 27](#_Toc37685151)

[8.10 Table CartonsJaunesHistory 28](#_Toc37685152)

[8.10.1 Triggers : 28](#_Toc37685153)

[8.10.2 Procédures stockées : 28](#_Toc37685154)

[8.11 Table GoalsHistory 29](#_Toc37685155)

[8.11.1 Triggers : 29](#_Toc37685156)

[8.11.2 Procédures stockées : 29](#_Toc37685157)

[8.12 Table FeuillesDeMatch 30](#_Toc37685158)

[8.12.1 Triggers : 30](#_Toc37685159)

[8.12.2 Procédures stockées : 30](#_Toc37685160)

[8.13 Table JoueursParticipationHistory 31](#_Toc37685161)

[8.13.1 Triggers 31](#_Toc37685162)

[8.13.2 Procédures stockées 32](#_Toc37685163)

[8.14 Table TransfertsHistory 33](#_Toc37685164)

[8.14.1 Triggers : 33](#_Toc37685165)

[8.14.2 Procédures stockées 34](#_Toc37685166)

[9 Mockups 35](#_Toc37685167)

[9.1 BackEnd 35](#_Toc37685168)

[9.1.1 Accueil 35](#_Toc37685169)

[9.1.2 Générer un championnat 36](#_Toc37685170)

[9.1.3 Calendrier des matchs 37](#_Toc37685171)

[9.1.4 Transférer des Joueurs 38](#_Toc37685172)

[9.1.5 Visualiser un match / le modifier 39](#_Toc37685173)

[9.1.6 Classement des équipes 39](#_Toc37685174)

[9.1.7 Classement des joueurs 40](#_Toc37685175)

[9.2 MatchManagement 41](#_Toc37685176)

[9.2.1 Accueil 41](#_Toc37685177)

[9.2.2 Feuilledematch 42](#_Toc37685178)

[9.2.3 Inscription des scores 43](#_Toc37685179)

[10 Listes erreurs 44](#_Toc37685180)

[10.1 TechnicalErrors – SQLExceptions 44](#_Toc37685181)

[10.2 BusinessErrors 45](#_Toc37685182)

[11 Liste des règles et contrôles de la solution 47](#_Toc37685183)

[12 Lessons learned 50](#_Toc37685184)

[12.1.1 Ne rien faire à l’avance 50](#_Toc37685185)

[12.1.2 Bien identifier la technologie nécessaire 50](#_Toc37685186)

[12.1.3 Découper son code et tester sur des petits morceaux de code 50](#_Toc37685187)

[12.1.4 DATAGRIDVIEW – DATATABLE - Dataview 51](#_Toc37685188)

[12.1.5 Le mieux est l’ennemi du bien 51](#_Toc37685189)

[13 Conclusion 52](#_Toc37685190)

# Contexte

## Cadre

Ce projet est développé dans le cadre du cours de Projet de développement SGBD de Monsieur Fievez.

A travers ce projet plusieurs points sont demandés à l’étudiant :

* Analyse d’une demande d’un client (pas toujours claire et précise)
* Réalisation d’une entité association
* Réalisation d’une base de données
* Réalisation de triggers et procédures stockées pour appliquer les contraintes
* Réalisation d’un code en 3 couches – DAL / Business / View
* Utilisations de modèles et d’erreurs séparés de la couche business
* Accès aux données en lecture et écriture uniquement par procédures stockées

D’autres points mériteraient également notre attention mais l’idée générale est dessinée : l’étudiant doit prouver qu’il est capable de développer une solution à un problème énoncé de manière non précise et changeante.

Des choix techniques, méthodologiques et analytiques doivent être réalisés par l’étudiant et doivent être justifiés au lecteur.

## Projet

Le projet en lui-même consiste au développement d’un système d’information permettant de faire fonctionner deux systèmes applicatifs avec des fonctionnalités différentes :

* BackEnd
* MatchManagement

Ces deux systèmes doivent utiliser une même base de données mais des utilisateurs et vue différentes.

# Analyse métier

## Description de la solution envisagée

La solution doit permettre la gestion de championnats de foot de la création du championnat en tant que tel à la gestion du calendrier et des résultats des matchs, des transferts de joueurs et de la lecture du classement.

Plus précisément, et comme évoqué plus haut, la solution est découpée en 2 parties distinctes.

Une partie BackEnd qui s’occupera principalement de la gestion du championnat (création, inscription équipe, génération du calendrier des matchs, transfert des joueurs, classements).

Une partie MatchManagement qui s’occupera de la gestion des matchs en tant que telle, inscription des joueurs sur la feuille de match, inscriptions des goals, cartons jaunes et rouges.

## Intervenants

Deux intervenants existeront pour notre solution :

* Un utilisateur BackEnd pour la partie BackEnd
* Un utilisateur MatchManagement pour la partie MatchManagement

# Fonctions attendues

## Exigences fonctionnelles

### BackEnd

La partie BackEnd devra permettre :

|  |  |
| --- | --- |
| FM-BE001 | La gestion des championnats |
| FM-BE002 | La gestion du calendrier des matchs |
| FM-BE003 | La gestion du transfert de joueurs |
| FM-BE004 | La gestion des matchs |
| FM-BE005 | La visualisation des classements par championnat |

### MatchManagement

La partie MatchManagement devra permettre :

|  |  |
| --- | --- |
| FM-MM001 | La gestion des feuilles de match |
| FM-MM002 | La gestion des matchs individuellement |

## Exigences non fonctionnelles

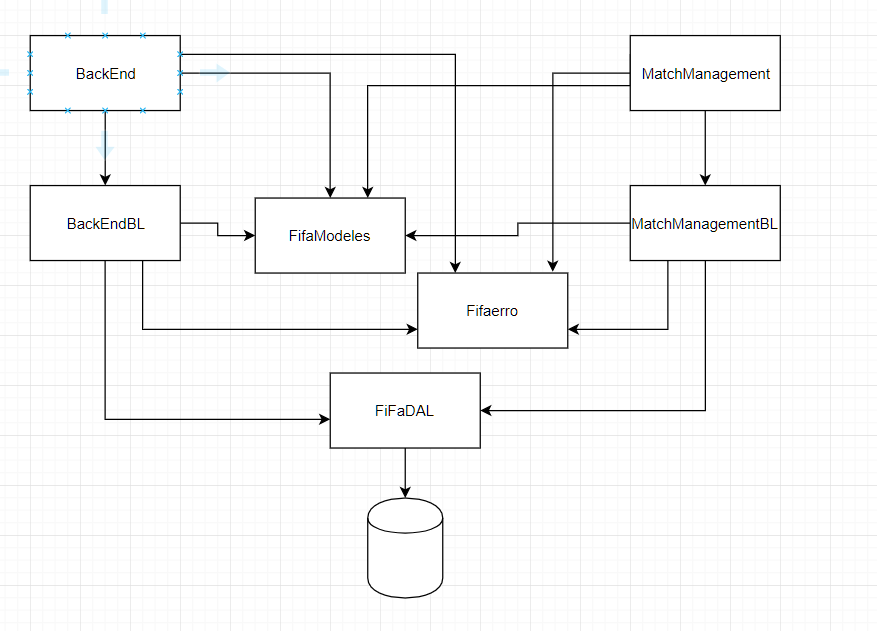
Le système d’information devra :

|  |  |
| --- | --- |
| FM-ENF001 | Être ergonomique pour l’utilisateur |
| FM-MM002 | Être Sécurisé |
| FM-MM003 | Permettre une vue complète et simplifiée des informations |

# Contraintes Business

La solution proposée devra :

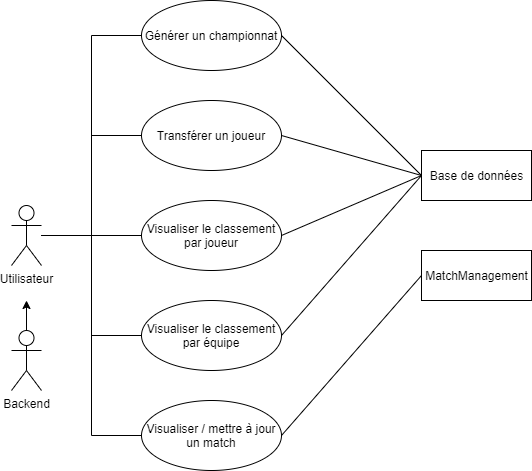
|  |  |
| --- | --- |
| FM-CB001 | Avoir une découpe en 3 couches (DAL, Business Layer, GUI) |
| FM-CB002 | Être divisé en 2 parties (BackEnd, MatchManahement) |
| FM-CB003 | Avoir une solution avec 7 projets (2 windows forms & 5 class libraries) et suivre le schéma ci-dessous |
| FM-CB004 | Accéder à la DB pour la partie BackEnd via Entity Framework DB First |
| FM-CB005 | Accéder à la DB pour la partie MatchManagement via ADO.NET |
| FM-CB006 | Avoir une seule base de données dédiée pour les 2 parties |
| FM-CB007 | Avoir un accès aux tables de la DB uniquement par l’utilisateur sa |
| FM-CB008 | Avoir 2 schémas différents (BackEnd et MatchManagement) |
| FM-CB009 | Avoir le schéma BackEnd accessible uniquement par sa et BackEnd |
| FM-CB010 | Avoir le schéma MatchManagement accessible uniquement par sa et MatchManagement |
| FM-CB011 | Accéder aux données et les modifier via des procédures stockées spécifiques par schéma. |



# Analyse fonctionnelle

## Diagrammes de Use Cases

### BackEnd



#### Use Case : Générer un championnat

##### Description textuelle ecran 1 – créer un championnat

Résumé : permet à l’utilisateur BackEnd de créer un championnat, inscrire les équipes éligibles dedans et fixer les dates des matchs

Acteurs : BackEnd *(primaire)* et la base de données *(secondaire)*

Préconditions :

* L’application BackEnd est lancée
* L’utilisateur a cliqué sur le bouton « Générer un championnat »
* La base de données existe et est valide (accès, structure, …)

Scénario nominal :

1. L’utilisateur encode l’année du championnat qu’il souhaite
2. Le logiciel vérifie que la date est valide (postérieur ou égale à 1900, inférieure ou égale à 9998) avec un format en quatre chiffres et s’il n’existe pas déjà un championnat pour cette année
3. Le logiciel calcule la date de début maximum du championnat
4. Le logiciel met à jour le datetime picker en limitant le choix de l’utilisateur
5. Le logiciel met à jour le résumé des dates en fonction de la date encodée dans le datetime picker
6. Le logiciel interroge la base de données pour savoir quelles équipes sont éligibles pour l’année encodée
7. La base de données répond avec la liste des équipes éligibles
   1. Une équipe doit posséder de 5 à 10 joueurs au début du championnat
8. Le logiciel affiche les équipes éligibles avec des cases sélectionnables
9. L’utilisateur sélectionne la date de début du championnat
10. Le logiciel met à jour le résumé
11. L’utilisateur sélectionne les équipes
12. L’utilisateur clique sur le bouton enregistrer et passer aux matchs
13. Le logiciel vérifie que le nombre d’équipe soit de minimum 2
14. Le logiciel interroge la base de données pour vérifier que le championnat n’existe pas déjà
15. Le logiciel crée le championnat dans la base de données
16. Le logiciel crée les quarters dans la base de données
17. Le logiciel crée l’intersaison dans la base de données
18. Le logiciel inscrit la participation des équipes au championnat dans la base de données
19. Le logiciel ouvre le second écran pour l’inscription des matchs.

Scénarios alternatifs :

*A1 :* L’utilisateur encode une année avec une étendue non-conforme (<1900 ou >9998) ou des caractères

1. Le logiciel affiche un message d’erreur
2. L’utilisateur recommence au point 1 du scénario nominal

*A2 :* L’utilisateur encode une année pour laquelle il y a déjà un championnat

1. Le logiciel affiche un message d’erreur
2. L’utilisateur recommence au point 1 du scénario nominal

*A3 :* L’utilisateur sélectionne moins de 2 équipes

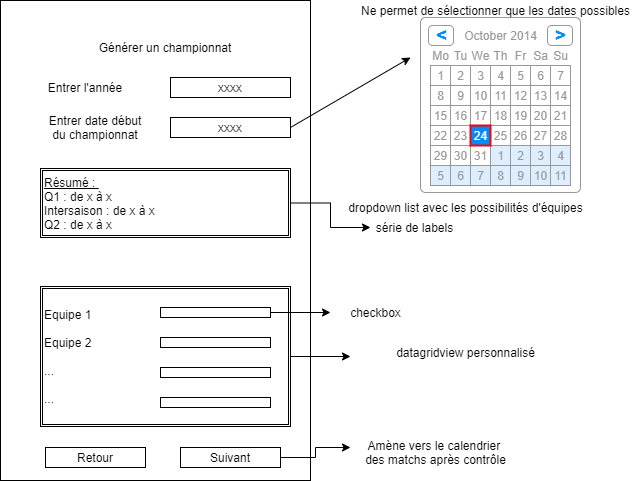
1. Le logiciel affiche un message d’erreur
2. L’utilisateur recommence au point 11 du scénario nominal

Enchainements d’erreurs :

*E1 :* L’utilisateur annule la création d’un championnat en cliquant sur le bouton « retour »

1. Le logiciel ferme la fenêtre et n’enregistre rien

##### Mockup ecran 1 – créer un championnat



##### Description textuelle ecran 2 – générer le calendrier des matchs

Résumé : permet à l’utilisateur BackEnd de contrôler les dates affectées aux matchs d’un championnat et de donner des dates aux matchs qui n’en ont pas encore, ou de modifier les existantes dans la limite des contraintes.

Acteurs : BackEnd *(primaire)* et la base de données *(secondaire)*

Préconditions :

* L’application BackEnd est lancée
* L’utilisateur vient de finir de créer un championnat via l’écran « générer un championnat »
* L’utilisateur a cliqué sur le bouton « Enregistrer »
* Toutes les conditions précédentes ont été remplies
* La base de données existe et est valide (accès, structure, …)

Scénario nominal :

1. Le logiciel génère automatiquement la liste des matchs et leur donne des dates selon les contraintes client et les choix de l’utilisateur lors de la création du championnat (année, liste des équipes, dates des quarters) :
   1. Chaque équipe joue une fois à domicile et une fois à l’extérieur contre toutes les autres équipes inscrites au championnat
   2. Une équipe ne peut pas joueur 2 fois sur un même week-end
   3. Une équipe ne peut pas jouer 2 fois le même jour, ni sans jour d’écart entre 2 matchs
   4. Les matchs d’une équipe doivent être répartis entre les 2 quarters
2. L’utilisateur vérifie que les dates lui conviennent bien et les modifie au besoin
3. L’utilisateur clique sur le bouton « sauvegarder »
4. Le logiciel crée les matchs dans la base de données.

Scénarios alternatifs :

*A1 :* Il y a trop d’équipes par rapport au nombre de week-end

1. Le logiciel indique comme date le 1er janvier 1801 pour les matchs du premier quarter qui n’ont pas pu recevoir de dates et le 1er janvier 1802 pour ceux du 2e quarter.
2. L’utilisateur modifie ces dates et recommence le scénario au point 3

*A2 :* L’utilisateur souhaite de l’aide pour trouver les lignes qu’il doit remplir

1. L’utilisateur clique sur le bouton aide
2. Le logiciel affiche en jaune toutes les lignes pour lesquelles une date valide n’a pas été donnée
3. Le scénario reprend au point 2 du scénario A1

Enchainements d’erreurs :

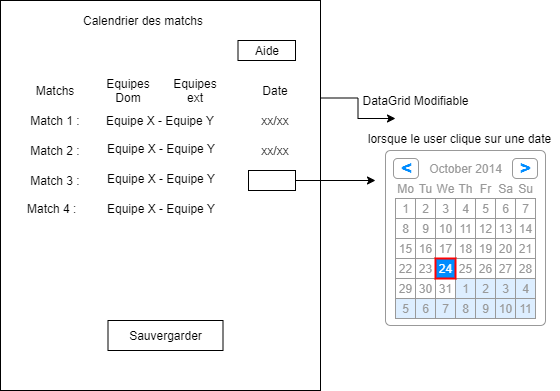
*E1 :* L’utilisateur n’a pas donné de date pour une ou plusieurs lignes qui le nécessitait

1. Le logiciel affiche un message d’erreur
2. Le logiciel affiche en jaune la ligne qui pose problème
3. Le scénario nominal reprend au point 2

*E2 :* L’utilisateur n’enregistre pas les matchs et ferme la fenêtre

1. Le logiciel affiche un message d’information
2. Le logiciel supprime le championnat précédemment crée dans la base de données
3. Le logiciel supprime les quarters précédemment crées dans la base de données
4. Le logiciel supprime l’intersaison précédemment crée dans la base de données

##### Mockup ecran 2 – générer le calendrier des matchs



Ne sont affichés que les matchs pour le championnat généré sur la form précédente.

Les matchs sont divisés entre les 2 quarters de manière « équitable » (une équipe ne fait pas tous ses matchs à domicile en un quarter mais ils sont distribués entre les 2)

Les dates de matchs sont générées automatiquement pour autant qu’une équipe ne joue qu’une fois sur le weekend. Les matchs supplémentaires ont pour date par défaut 1/1/1801 pour le premier quarter et 1/1/1802 pour le 2ème quarter.

Dans tous les cas, une équipe ne peut pas jouer 2 matchs le même jour ou 2 jours de suite.

#### Use Case : Transférer un joueur

##### Description textuelle– Transférer un joueur

Résumé : permet à l’utilisateur BackEnd de transférer des joueurs d’un club à l’autre

Acteurs : BackEnd *(primaire)* et la base de données *(secondaire)*

Préconditions :

* L’application BackEnd est lancée
* L’utilisateur a cliqué sur le bouton « Transférer un joueur »
* La base de données existe et est valide (accès, structure, …)

Scénario nominal :

1. Le logiciel demande à la base de données toutes les informations de transferts de joueurs.
2. La base de données renvoie les informations
3. Le logiciel affiche les données nécessaires pour l’utilisateur
4. L’utilisateur encode ses transferts
5. L’utilisateur appui sur le bouton « enregistrer et rafraichir »
6. Le logiciel vérifie que les dates remplissent toutes les conditions
   1. Une date de transfert doit être encodée pour qu’un transfert ait lieu
   2. Le joueur ne peut pas avoir participé à un match à une date ultérieure
   3. La date de transfert ne doit pas être comprise dans un quarter
   4. Vérifie que les équipes d’arrivée et de départ remplissent toujours les conditions de nombre de joueurs
   5. Vérifie si le transfert a lieu lors d’une intersaison que l’équipe de destination fasse bien partie des 3 derniers du classement
7. Le logiciel clôture l’ancien transfert s’il existait dans la base de données
8. Le logiciel enregistre le nouveau transfert s’il y en a un.
9. Le logiciel demande à la base de données toutes les informations de transferts de joueurs.
10. La base de données renvoie les informations
11. Le logiciel affiche les données nécessaires pour l’utilisateur

Il s’agit ici de traduire la compréhension de l’énoncé initial en des termes clairs et précis.

L’objectif n’est pas d’avoir l’entièreté des contraintes expliquées, uniquement celles de l’énoncé.

Il s’agit ici, de la base, du socle qui me permettra de proposer une solution développée au problème soumis dans le cadre du cours.

Par la suite, les contraintes « dérivées » de celles-ci seront expliquées.

## Championnats

Un championnat se déroule sur une année civile.

Il est divisé en 2 quarters de 5 semaines séparés par une intersaison de 6 mois.

Le nombre d’équipes inscrites varie d’une année à l’autre.

Pour pouvoir s’inscrire dans un championnat une équipe doit être composée au minimum de 5 joueurs.

## Équipes

Une équipe est composée de 5 à 10 joueurs et possède un nom et peut avoir un logo

Une équipe peut s’inscrire à des championnats.

## Joueurs

Un joueur possède un nom, un prénom

Un joueur est inscrit ou non dans une équipe

Il peut posséder plusieurs cartes jaunes et au maximum une carte rouge.

S’il possède une carte rouge active, il ne peut pas disputer les 3 prochains matchs

S’il possède le même nombre ou plus de cartes jaunes actives que de matchs restants dans l’intersaison, il ne peut plus jouer pendant le quarter.

S’il manque un match et :

* Qu’il a un carton rouge avec des suspensions
* Qu’il n’a pas de carton rouge avec des suspensions et qu’il a un ou plusieurs carton jaunes actifs, le carton jaune le plus ancien est désactivé
* Qu’il n’ait ni carton rouge avec suspensions ni un carton jaune actif alors rien ne se passe

À la fin d’un quarter tous les cartons jaunes sont désactivés et les suspensions pour les cartons rouges sont remises à 0.

Un joueur ne peut pas changer d’équipe lors d’un quarter.

Un joueur ne peut changer d’équipe à l’intersaison que si son équipe possède plus de 5 joueurs et que l’équipe dans laquelle il va est classée parmi les 3 dernières du championnat.

Un joueur peut changer d’équipe entre 2 championnats.

## Matchs

Les matchs ne se jouent que le samedi ou le dimanche sauf s’il y a trop de match pour le nombre de week-end. Une équipe ne joue qu’une fois par week-end.

Un match est joué par 2 équipes. Chaque équipe remplit une feuille de match.

Si la feuille de match d’une équipe compte moins de 5 joueurs et :

* Moins de joueurs que celle de l’adversaire alors elle perd automatiquement le match.
* Le même nombre de joueur que celle de l’adversaire alors c’est un match nul.

Un match possède un résultat et donne un certain nombre de points par équipe :

* Gagné : 3 points
* Perdu : 0 point
* Nul : 1 point

Lors d’un match, un joueur peut marquer un goal ou prendre un carton jaune ou rouge.

## Feuilles de match

Une feuille de match peut compter 7 joueurs maximum et est unique par match et par équipe.

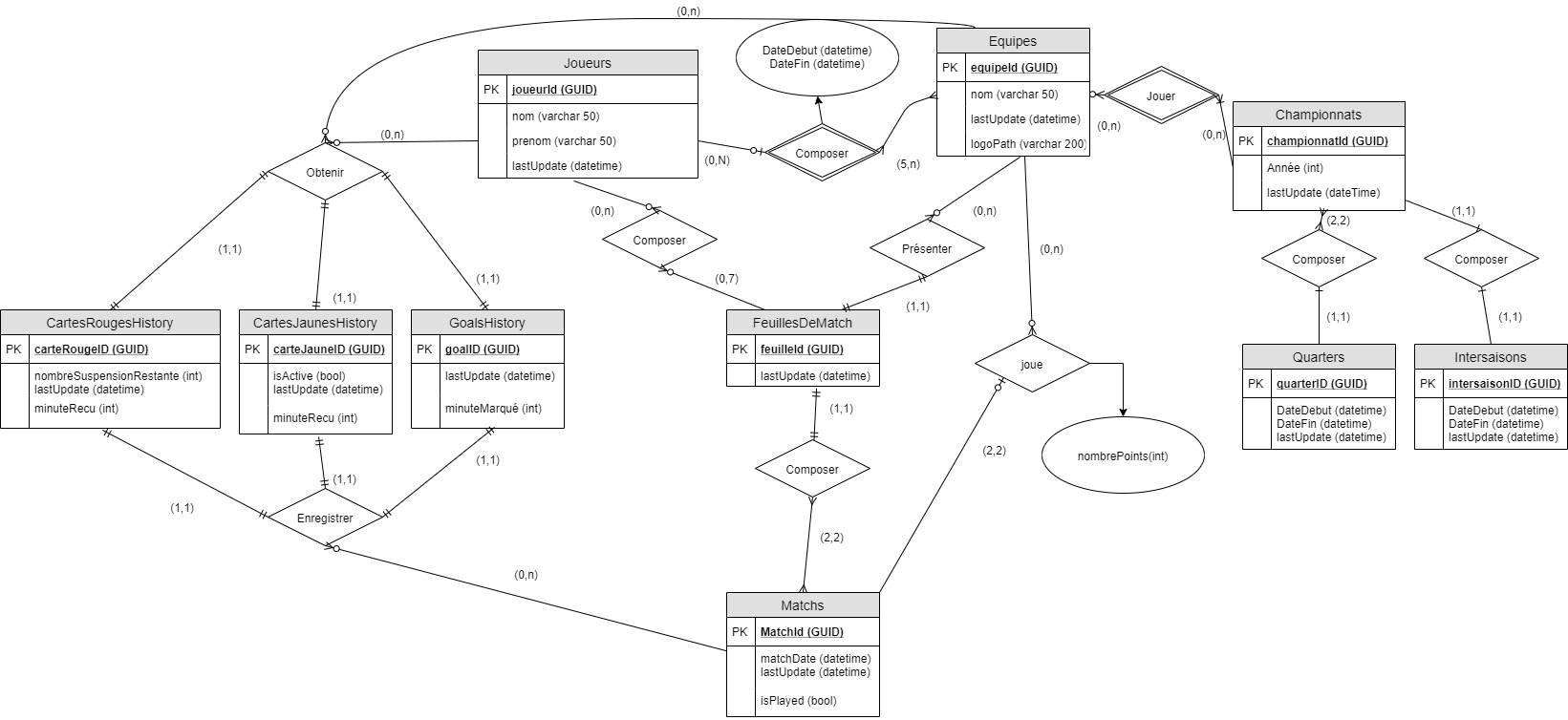
Un joueur ne peut être inscrit que s’il est dans l’équipe.

Un joueur qui a une carte rouge active ne peut pas être inscrit sur la feuille de match.

Si un joueur à le même nombre de carte jaune que de match restant à jouer pour son équipe, il ne peut pas être inscrit sur la feuille de match.

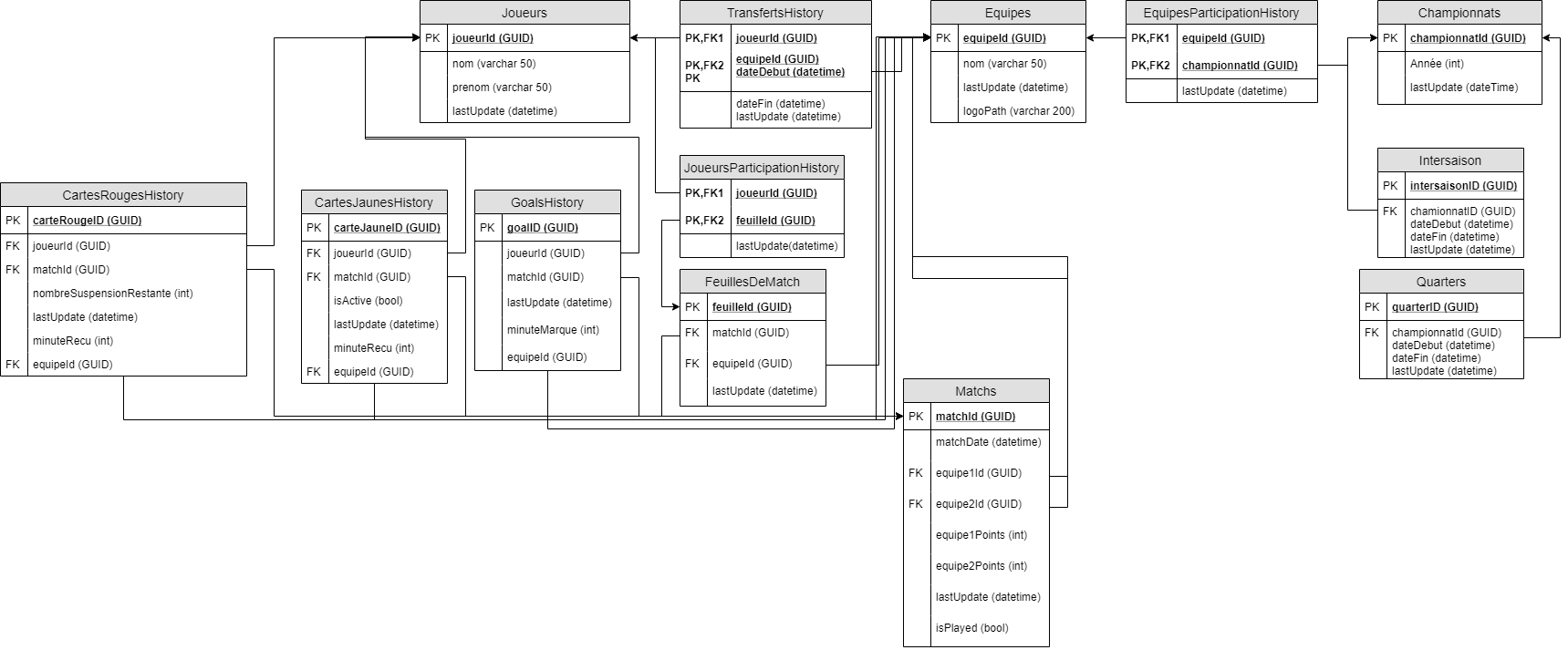
# Entité Association

Vous trouverez ci-dessous l’entité association proposée pour ce projet.



# Schéma Relationnel

Vous trouverez ci-dessous le schéma relationnel proposé pour ce projet. La description de chaque table faisant l’objet du prochain chapitre.



# Tables SQL

Nous allons ici parcourir l’ensemble des tables du schéma relationnel et établie l’ensemble des contraintes, des procédures stockées et des triggers qui y sont attachés.

La lecture de cette partie, technique, sera sans doute laborieuse pour le lecteur non averti mais permettra de définir dans son entièreté la base de donnée pour la solution proposée.

## Avant-Propos

Certaines procédures stockées citées ci-dessous testent des conditions qui sont déjà vérifiées dans les contraintes de la base de données. Cette étape a pour but de fournir des messages d’erreurs personnalisés.

L’ensemble des erreurs retournées se trouvent au chapitre du même nom.

Certaines conditions sont vérifiées aussi bien lors d’un trigger que dans une procédure stockée. Il s’agit d’un choix délibéré de tester ces conditions business à tous les étages.

Certaines tables ne comptent pas d’attribut d’ID, il s’agit d’un également choix délibéré. S’agissant d’un travail d’ordre scolaire, l’efficacité n’a pas toujours été priorisée, il s’agit parfois de démontrer l’usage de certaines techniques et maitrises du langage SQL.

De plus, tel que prévu par l’énoncé, 2 schémas et utilisateurs attachés à la base de données ont été créés :

* BackEnd
* MatchManagement

## Table Championnat

Une image contenant capture d’écran

Description générée automatiquement

La table Championnat comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté automatiquement à la création de tout nouveau championnat.

Aucun de ses champs ne peut être nul. Le champ Année est unique en INT ; pour éviter les erreurs d’encodage, il est décidé que l’année doit être supérieur ou égale à 1900 ou un rollback transaction est réalisé

lastUpdate est utilisé pour pouvoir faire un suivi, il n’est pas prévu de modifier un championnat.

### Trigger :

* FormatAnneesChampionnats : vérifie que l’année est supérieure ou égale à 1900 lors d’un insert ou d’un update.

### Procédures stockées :

* BackEnd.Championnats\_Add (@année int) : rajoute un championnat à condition qu’il n’y ait pas de championnat avec la même année et que l’année est supérieure ou égale à 1900.
* BackEnd et MatchManagement.Championnats\_GetAll : retourne tous les championnats

## Table Intersaisons



La table Intersaison comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouvelle intersaison

Elle contient également une date de début et une date de fin en datetime et une clé étrangère vers la table Championnats qui est unique. Aucun de ses champs ne peut être nul.

lastUpdate est utilisé pour pouvoir faire un suivi, il n’est pas prévu de modifier une intersaison

### Trigger :

* Intersaisons\_UniciteChampionnat : vérifie qu’il n’y ait bien qu’une intersaison par championnat

### Procédures stockées :

* BackEnd.CreationDivisionChampionnat (@DateDebutChampionnat DATETIME, @ChampionatId uniqueidentifier) : crée 2 quarters et une intersaison à partir d’un championnat et d’une date de début de championnat. Vérifie l’existence du championnat, que la fin du championnat finisse bien dans la même année que le championnat et qu’il n’existe pas déjà de quarters et d’intersaison pour ce championnat.
* BackEnd.Intersaisons\_GetAll : renvoie toutes les intersaisons
* BackEnd.Intersaisons.Add : (@dateDebut DATETIME,@dateFin DATETIME,@ChampionnatId uniqueidentifier) vérifie si il n’existe pas déjà une intersaison pour ce championnat, que la date de début et de fin sont dans la même année et que l’année de date de début corresponde à l’année du championnat et rajoute une intersaison

## Table Quarters



La table Quarters comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouveau quarter

Elle contient également une date de début et une date de fin en datetime et une clé étrangère vers la table Championnats qui ne peut être reprise que 2 fois à l’identique.

Aucun de ses champs ne peut être nul.

lastUpdate est utilisé pour pouvoir faire un suivi, il n’est pas prévu de modifier un quarter

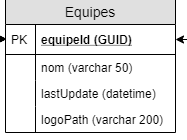
### Triggers :

* MaxQuarters : vérifie qu’il n’existe pas déjà 2 quarters pour un championnat.

### Procédures stockées :

* BackEnd.CreationDivisionChampionnat (@DateDebutChampionnat DATETIME, @ChampionatId uniqueidentifier) : crée 2 quarters et une intersaison à partir d’un championnat et d’une date de début de championnat. Vérifie l’existence du championnat, que la fin du championnat finisse bien dans la même année que le championnat et qu’il n’existe pas déjà de quarters et d’intersaison pour ce championnat.
* MatchManagement et BackEnd.Quarters\_GetAll : renvoie tous les quarters
* BackEnd.Quarters.Add : (@dateDebut DATETIME,@dateFin DATETIME,@ChampionnatId uniqueidentifier) vérifie si il n’existe pas déjà une intersaison pour ce championnat, que la date de début et de fin sont dans la même année et que l’année de date de début corresponde à l’année du championnat et rajoute un quarter

## Table Equipes



La table Equipes comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouvelle équipe

Elle contient un nom qui est unique et un logoPath qui donne l’accès au logo de l’équipe. Par défaut ce path est noLogo. Aucun de ses champs ne peut être nul.

lastUpdate est utilisé pour pouvoir faire un suivi, il n’est pas prévu de créer ou de modifier une équipe via nos programmes.

### Triggers :

* Equipes\_UniciteNom : vérifie que le nom d’équipe n’est pas déjà utilisé.

### Procédures stockées :

* MatchManagement.Equipes\_GetAll : renvoie toutes les équipes

## Table EquipesParticipationHistory

Une image contenant capture d’écran

Description générée automatiquement

La table EquipesParticipationHistory comprend les Id en GUID de chaque équipe et du/des championnat(s) auxquels ils ont participé ainsi qu’un lastUpdate en datetime qui est implémenté à la création de toute nouvelle participation[[1]](#footnote-1).

Une équipe ne peut participer qu’une fois à un championnat. Une équipe doit avoir assez de joueurs pour s’inscrire au championnat.

Aucun de ses champs ne peut être nul.

lastUpdate est utilisé pour pouvoir faire un suivi, il n’est pas nécessaire de vérifier si on travaille sur des données plus anciennes que celles de la bdd. Une équipe participe ou non à un championnat.

### Trigger :

* EquipesParticipation\_EquipeActive : vérifie que l’équipe réponde aux conditions d’inscription (5 joueurs mini) au championnat avant de l’inscrire.

### Procédures stockées :

* BackEnd.EquipeParticipation\_Add (@equipeId uniqueidentifier, @championnatId uniqueidentifier) : vérifie :
  + que le championnat et l’équipe existe ;
  + que la combinaison n’existe pas déjà ;
  + que l’équipe au début du championnat est composée d’au moins 5 joueurs.

Et dès lors crée l’association de participation.

* BackEnd.EquipeParticipation\_GetAll : renvoie toutes les participation des équipes

## Table Joueurs

Une image contenant capture d’écran

Description générée automatiquement

La table Joueurs comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouveau joueur. Un joueur possède également un nom et un prénom en varchar (50). Aucun de ses champs ne peut être nul.

lastUpdate est utilisé pour pouvoir faire un suivi. Il n’est pas prévu de créer ou de modifier un joueur

### Procédures stockées

* BackEnd et MatchManagement.Joueurs\_GetAll : renvoie tous les joueurs

## Table Matchs

Une image contenant capture d’écran

Description générée automatiquement

La table Matchs comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime, des points pour les 2 équipes initialisés à 0 en INT et un bit pour savoir si le match a été joué (0 par défaut) qui sont implémentés à la création de tout nouveau match.

Un match possède également une date de rencontre en datetime (qui doit être compris dans un quarter) et 2 clés étrangères qui sont 2 équipes distinctes. De plus la combinaison équipe 1 – équipe 2 doit être unique lors d’un championnat.

Aucun de ses champs ne peut être nul.

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’un match des données plus récentes n’existent pas suite à un délai d’encodage.

La table est mise à jour automatiquement lors des insertions et modifications de cartes jaunes, rouges et de goals. Il est considéré qu’un match est joué dès qu’on commence à encoder un de ces 3 points. Si le compte des goals pour ce match est nul des 2 côtés et qu’on inscrit un carton alors les points sont automatiquement mis à 1 pour les 2 équipes.

La table est également mise à jour lorsqu’une feuille de match est remplie. Si une des 2 feuilles de match compte moins de 5 joueurs alors l’autre équipe gagne automatiquement. Si les 2 équipes ont moins de 5 joueurs sur leur feuille de match, celle avec le plus de joueur gagne le match.

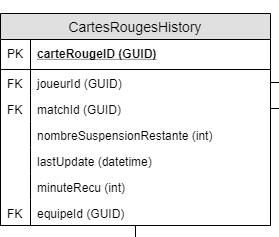
### Triggers :

* Matchs\_WithinChampionnat : vérifie que les matchs créés ou modifiés ont bien une date qui correspond à un quarter existant.
* Matchs\_SameTeams : vérifie que les 2 équipes d’un match ne soient pas les mêmes
* Matchs\_Doublons : vérifie qu’un match n’a pas déjà été prévu entre ses 2 équipes, dans le même sens et dans le même championnat.
* Matchs\_UpdatePlusVieux : vérifie qu’une mise à jour des données n’a pas été réalisée après l’encodage.
* CartonsRougesHistory\_PlayedMatch : Indique un match comme joué dès qu’il y a une insertion d’un carton rouge et mets à jour le match (lastUpdate) si les points des 2 équipes ne sont pas encore initialisés, les mets à 1 pour chaque équipe
* CartonsJaunesHistory\_PlayedMatch : Indique un match comme joué dès qu’il y a une insertion d’un carton jaune et mets à jour le match (lastUpdate) si les points des 2 équipes ne sont pas encore initialisés, les mets à 1 pour chaque équipe
* GoalsHistory\_PlayedMatch : Indique un match comme joué dès qu’il y a une insertion d’un goal et mets à jour le match (lastUpdate) et compte les points de chaque équipe
* JoueursParticipationHistory\_CheckFeuillesDeMatch : vérifie le nombre de joueurs inscrits pour le match sur les 2 feuilles de match et attribue des points en fonction de la complétude ou non des feuilles. Si une des 2 feuilles de match n’a pas 5 joueurs le match est joué et est considéré comme un forfait sauf dans le cas où les 2 équipes ont le même nombre de joueurs inscrit, ce qui donne un nul.

### Procédures stockées :

* BackEnd.Matchs\_Add (datetime @matchDate, uniqueidentifier @equipe1Id, uniqueidentifier @equipe2Id) : rajoute un match en vérifiant les mêmes conditions que les 3 premiers triggers sur la table.
* BackEnd et MatchManagement.Matchs\_Update (@matchId uniqueidentifier, @matchDate datetime, @equipe1Id uniqueidentifier, @equipe2Id uniqueidentifier, @equipe1Points int, @equipe2Points int,@lastUpdate datetime) mets à jour un match en vérifiant les 4 conditions.
* BackEnd et MatchManagement.Matchs\_GetAll : renvoie tous les matchs

## Table CartonsRougesHistory



La table CartonsRougesHistory comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouveau carton rouge. Elle possède 3 clés étrangères : joueurId, equipeId et matchId. NombreSuspensionRestante et minuteRecu sont des int.

L’attribut minuteRecu doit être inclus entre 0 et 120, sans prolongations un match dure 90 minutes avec les prolongations 120 minutes. Un joueur ne peut recevoir qu’un carton rouge par match.

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’une carte rouge des données plus récentes n’existent pas suite à un délai d’encodage.

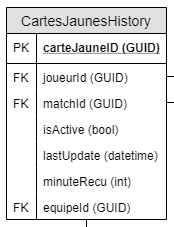
### Triggers :

* CartonsRougesHistory\_LimiteMinuteRecue : vérifie que le carton rouge est obtenu entre 0 et 120 minutes de match
* CartonsRougesHistory\_UpdatePlusVieux : vérifie qu’une mise à jour des données n’a pas été réalisée après l’encodage.
* CartonsRougesHistory\_MaxCartonParMatch : vérifie qu’un joueur ne reçoive pas plusieurs cartes rouges lors d’un même match
* CartonsRougesHistory\_EstDansFeuilleDeMatch : vérifie que le joueur joue bien le match.
* CartonsRougesHistory\_PlayedMatch : Indique un match comme joué dès qu’il y a une insertion d’un carton rouge et mets à jour le match (lastUpdate) si les points des 2 équipes ne sont pas encore initialisés, les mets à 1 pour chaque équipe
* CartonsRougesHistory\_EnoughJoueurs : vérifie que les 2 feuilles de match comprennent au minimum 5 joueurs avant d’inscrire un carton rouge

### Procédures stockées :

* BackEnd et MatchManagement.CartonsRougesHistory\_Add (@joueurId uniqueidentifier, @matchId uniqueidentifier, @nombreSuspensionsRestantes INT, @minuteRecue INT) : ajoute un carton rouge à un joueur à condition qu’il n’en ait pas déjà reçu un dans le même match
* BackEnd et MatchManagement.CartonsRouges\_DiminueSuspension (@carteRougeId uniqueidentifier) : diminue la suspension restante d’une carte rouge de 1.
* BackEnd et MatchManagement.CartonsRouges\_Update (@carteRougeId uniqueidentifier, @joueurId uniqueidentifier, @matchId uniqueidentifier, @nombreSuspensionsRestantes INT, @minuteRecue INT, @lastUpdate datetime) : met à jour une carte existante et vérifie qu’un joueur ne reçoive qu’une carte rouge par match et que la carte modifiée existe bien.
* BackEnd Matchmanagement.CartonsRouges\_GetAll : renvoie toutes les cartes rouges et la date du match avec au moins une suspension
* BackEnd et MatchManagement.CartonsRouges\_Delete (@carteRougeId uniqueidentifier) : supprime une carte rouge

## Table CartonsJaunesHistory



La table CartonsJaunesHistory comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouveau carton jaune. Elle possède 3 clés étrangères : joueurId, equipeId et matchId. minuteRecu est un int et isActive est un bit initialisé par défaut à 1.

L’attribut minuteRecu doit être inclus entre 0 et 120, sans prolongations un match dure 90 minutes avec les prolongations 120 minutes.

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’une carte jaune des données plus récentes n’existent pas suite à un délai d’encodage.

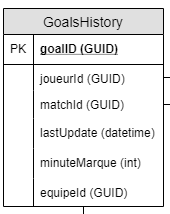
### Triggers :

* CartonsJaunesHistory\_LimiteMinuteRecue : vérifie que le carton jaune est obtenu entre 0 et 120 minutes de match
* CartonsJaunesHistory\_UpdatePlusVieux : vérifie qu’une mise à jour des données n’a pas été réalisée après l’encodage.
* CartonsJaunesHistory\_EstDansFeuilleDeMatch : vérifie que le joueur joue bien le match.
* CartonsJaunesHistory\_PlayedMatch : Indique un match comme joué dès qu’il y a une insertion d’un carton jaune et mets à jour le match (lastUpdate) si les points des 2 équipes ne sont pas encore initialisés, les mets à 1 pour chaque équipe
* CartonsJaunesHistory\_EnoughJoueurs : vérifie que les 2 feuilles de match comprennent au minimum 5 joueurs avant d’inscrire un carton jaune

### Procédures stockées :

* BackEnd et MatchManagement.CartonsJaunes\_Add (@joueurId uniqueidentifier, @matchId uniqueidentifier,@minuteRecue INT) : ajoute un carton jaune à un joueur lors d’un match
* BackEnd et MatchManagement.CartonsJaunes\_Update (@carteJauneId uniqueidentifier, @joueurId uniqueidentifier, @matchId uniqueidentifier, @minuteRecue INT, @lastUpdate datetime) : met à jour une carte existante et vérifie que la carte modifiée existe bien.
* MatchManagement.CartonsJaunes\_GetAll : renvoie tous les cartonsjaunesId, joueursId, isActive et matchDate
* MatchManagement.CartonsJaunes\_Delete : supprime une carte jaune

## Table GoalsHistory



La table GoalsHistory comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de tout nouveau goal.

Elle possède 3 clés étrangères : joueurId, equipeId et matchId. L’attribut minuteMarque est un int.

L’attribut minuteRecu doit être inclus entre 0 et 120, sans prolongations un match dure 90 minutes avec les prolongations 120 minutes.

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’une carte jaune des données plus récentes n’existent pas suite à un délai d’encodage.

### Triggers :

* GoalsHistory\_LimiteMinuteRecue : vérifie que le goal est marqué entre 0 et 120 minutes de match
* GoalsHistory\_UpdatePlusVieux : vérifie qu’une mise à jour des données n’a pas été réalisée après l’encodage.
* GoalsHistory\_EstDansFeuilleDeMatch : vérifie que le joueur joue bien le match.
* GoalsHistory\_PlayedMatch : Indique un match comme joué dès qu’il y a une insertion d’un goal et mets à jour le match (lastUpdate) et compte les points de chaque équipe
* GoalsHistory\_EnoughJoueurs : vérifie que les 2 feuilles de match comprennent au minimum 5 joueurs avant d’inscrire un goal

### Procédures stockées :

* BackEnd et MatchManagement.Goals\_Add (@joueurId uniqueidentifier, @matchId uniqueidentifier, @minuteMarque INT) : ajoute un goal à un joueur lors d’un match
* BackEnd et MatchManagement.Goals\_Update (@goalId uniqueidentifier, @joueurId uniqueidentifier, @matchId uniqueidentifier, @minuteMarque INT,@lastUpdate datetime) : met à jour un goal existant et vérifie que le goal modifié existe bien.
* backEnd et MatchManagement.Goals\_GetAll : renvoie tous les goals
* BackEnd et MatchManagement.Goals\_Delete @goalId uniqueidentifier : supprime un goal

## Table FeuillesDeMatch

Une image contenant capture d’écran

Description générée automatiquement

La table FeuillesDeMatch comprend un Id en GUID généré automatiquement ainsi qu’un lastUpdate en datetime qui est implémenté à la création de toute nouvelle feuille de match. Elle possède 2 clés étrangères : matchId et equipeId. Aucun de ses champs ne peut être nul.

Il ne peut y avoir qu’une feuille de match pour une équipe pendant un match précis.

Il ne peut avoir que 2 feuilles de match pour un même match.

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’une feuille de match des données plus récentes n’existent pas suite à un délai d’encodage

### Triggers :

* FeuillesDeMatch\_MaxFeuilleParEquipe : vérifie qu’il n’y ait pas plusieurs feuilles de match pour une même équipe lors d’un même match
* FeuillesDeMatch\_MaxFeuilleParMatch : vérifie qu’il n’y ait pas déjà 2 feuilles de match pour un même match
* FeuillesDeMatch\_UpdatePlusVieux : vérifie qu’une mise à jour des données n’a pas été réalisée après l’encodage.

### Procédures stockées :

* BackEnd et MatchManagement.FeuilleDeMatch\_Add (@matchId uniqueidentifier, @equipeId uniqueidentifier) : ajoute une feuille de match s’il n’y a pas déjà 2 feuilles pour ce match et s’il n’y a pas déjà une feuille pour cette équipe et ce match.
* BackEnd et MatchManagement.FeuilleDeMatch\_Update (@feuilleId uniqueidentifier, @matchId uniqueidentifier, @equipeId uniqueidentifier, @lastUpdate datetime) : modifie une feuille de match si elle existe et répond aux mêmes conditions que les 3 triggers ci-dessus si elle n’existe pas en crée une nouvelle.
* BackEnd et MatchManagement.FeuilleDeMatch\_GetAll : renvoie toutes les feuilles de match

## Table JoueursParticipationHistory

Une image contenant capture d’écran

Description générée automatiquement

La participation d’un joueur à une feuille de match étant unique, la clé primaire[[2]](#footnote-2) est composée de 2 clés étrangères : joueurId et feuilleId.

Un lastUpdate en datetime est implémenté à la création de toute nouvelle participation d’un joueur.

Aucun des champs de la table ne peut être nul. lastUpdate est utilisé pour pouvoir faire un suivi, soit on ajoute une participation, soit on la supprime.

Il ne peut avoir que 7 joueurs maximum par feuille de match.

Un joueur ne peut être inscrit que s’il joue dans l’équipe de la feuille de match.

Un joueur qui a une carte rouge active ne peut pas être inscrit sur la feuille de match.

Si un joueur à le même nombre de carte jaune que de match restant à jouer pour son équipe, il ne peut pas être inscrit sur la feuille de match.

### Triggers

* JoueursParticipationHistory\_MaxJoueursParFeuille : vérifie qu’il n’y ait pas plus de 7 joueurs inscrits pour une feuille de match
* JoueursParticipationHistory\_EstDansEquipe : vérifie que le joueur inscrit dans la feuille de match joue bien pour la bonne équipe
* JoueursParticipationHistory\_CartonRouge : vérifie qu’un joueur n’a pas de carton rouge avec des suspensions restantes dans les 6 semaines précédant le match.
* JoueursParticipationHistory\_CartonJaune : vérifie que le joueur inscrit n’a pas autant ou plus de cartons jaunes que de match restant pour son équipe
* JoueursParticipationHistory\_CheckFeuillesDeMatch : vérifie le nombre de joueurs inscrits pour le match sur les 2 feuilles de match et attribue des points en fonction de la complétude ou non des feuilles. Si une des 2 feuilles de match n’a pas 5 joueurs le match est joué et est considéré comme un forfait sauf dans le cas où les 2 équipes ont le même nombre de joueurs inscrit, ce qui donne un nul.

### Procédures stockées

* BackEnd et MatchManagement.JoueursParticipation\_Add (@joueurId uniqueidentifier, @feuilleId uniqueidentifier) : ajoute une participation d’un joueur à condition qu’il réponde aux conditions décrites ci-dessus.
* BackEnd et MatchManagement.JoueursParticipation\_GetOne (@feuilleId uniqueidentifier) : renvoie tous les joueurs sur la @feuilleId
* MatchManagement.JoueursParticipation\_Delete (@joueurId uniqueidentifier,@feuilleId uniqueidentifier) : supprime la participation
* BackEnd et MatchManagement.FeuilleDeMatch\_GetNombreJoueursInscrits (@matchId uniqueidentifier,@equipeId uniqueidentifier) : renvoie la table des joueurs inscrit sur la feuille de match d’une équipe.
* Backend et MatchManagement.JoueursParticipation\_GetAll : renvoie toutes les participations

## Table TransfertsHistory

Une image contenant capture d’écran

Description générée automatiquement

La table TransfertsHistory a comme clé primaire[[3]](#footnote-3) les 2 clés étrangères joueurId et equipeId ainsi que la dateDebut en datetime.

De plus, la table comprend la dateFin du transfert et un lastUpdate en datetime.

DateFin peut être nul.

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’une feuille de match des données plus récentes n’existent pas suite à un délai d’encodage

Une équipe est composée de 5 à 10 joueurs

Un joueur ne peut pas changer d’équipe lors d’un quarter.

Un joueur ne peut changer d’équipe à l’intersaison que si son équipe possède plus de 5 joueurs et que l’équipe dans laquelle il va est classée parmi les 3 dernières du championnat.

Un joueur peut changer d’équipe entre 2 championnats.

### Triggers :

* TransfertsHistory\_UpdatePlusVieux : vérifie qu’une mise à jour des données n’a pas été réalisée après l’encodage.
* TransfertsHistory\_NombreJoueurs : vérifie lors d’un update qu’il y ait entre 5 et 10 joueurs dans une équipe
* TransfertsHistory\_NombreJoueursMax : vérifie lors d’un insert qu’il n’y a pas plus de 10 joueurs dans une équipe
* TransfertsHistory\_DateTransfert : vérifie que le transfert n’ait pas lieu lors d’un quarter. S’il a lieu lors d’une intersaison il ne peut aller que dans une des 3 dernières équipes.
* TransfertHistory\_ActiveTeam : active une équipe si elle a entre 5 et 10 joueurs inscrits
* TransfertsHistory\_DeuxEquipes : vérifie qu’un joueur ne soit pas déjà dans une équipe avant de l’adjoindre à une équipe.

### Procédures stockées

* BackEnd.Tansferts\_Add ( @joueurId uniqueidentifier, @equipeId uniqueidentifier, @dateTransfert datetime,@lastUpdate datetime) : réalise un transfert de joueur, si il était déjà dans une équipe met également fin à la collaboration
* BackEnd et MatchManagement.Transferts\_GetParticipants @equipeId uniqueidentifier,@matchDate DATETIME) : renvoie les joueurs étant dans une équipe à une date précise.
* BackEnd.Transferts.GetAll : renvoie tous les transferts

# Mockups

L’exercice des mockups est assez éclairants sur les contraintes « dérivées » de notre problème.

En plus des contraintes imposées par l’énoncé, toute une série de contraintes va faire son apparition lorsque nous débuterons la réflexion de l’interface vers l’utilisateur, du comment rendre compréhensible et facile d’utilisation notre solution.

## BackEnd

Un MDI Form contient toutes les autres formes de BackEnd

### Accueil

Une image contenant capture d’écran

Description générée automatiquement

Menu de base présentant les différentes options offertes à l’utilisateur.

### Générer un championnat

En cliquant sur suivant le championnat et les quarters et l’intersaison sont créés si ils répondent à tous les critères (n’existe pas déjà, …)

Vérifie également avant d’enregistrer une équipe si elle n’a pas déjà eu des transferts encodés l’année qui va être encodée

### Calendrier des matchs

Une image contenant capture d’écran

Description générée automatiquement

Ne sont affichés que les matchs pour le championnat généré sur la form précédente.

Les matchs sont divisés entre les 2 quarters de manière « équitable » (une équipe ne fait pas tous ses matchs à domicile en un quarter mais ils sont distribués entre les 2)

Les dates de matchs sont générées automatiquement pour autant qu’une équipe ne joue qu’une fois sur le weekend. Les matchs supplémentaires ont pour date par défaut 1/1/1801 pour le premier quarter et 1/1/1802 pour le 2ème quarter.

Dans tous les cas, une équipe ne peut pas jouer 2 matchs le même jour ou 2 jours de suite.

Un match qui doit avoir lieu dans le premier quarter ne peut être mis qu’au premier quarter idem pour le deuxième quarter. Le datetimepicker s’assure du respect de cette règle.

Si il y a une business error qui s’enclenche la ligne qui pose problème est surlignée en jaune pour permettre à l’utilisateur de repérer plus facilement la ligne qui pose problème.

### Transférer des Joueurs

Une image contenant capture d’écran

Description générée automatiquement

La date de transfert est au minimum un jour après la date in s’il y en a une sinon, elle est du 1/1/1900.

Lors d’un choix de date, il est déjà vérifié que la date choisie n’est pas lors d’un quarter ou que le joueur n’est pas déjà inscrit sur une feuille de match à une date postérieure.

Pour rappel, une équipe ne peut être inscrite à un championnat si elle a déjà un transfert encodé la même année que celle du championnat.

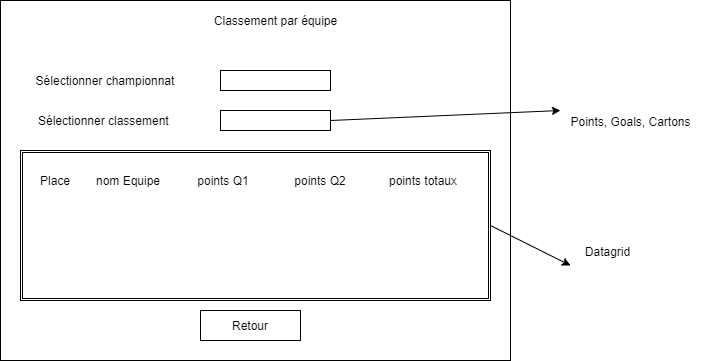
Avant d’enregistrer les transferts, il est vérifié si la date est lors d’une intersaison, si oui il est vérifié :

* si l’équipe d’arrivée est inscrite dans le championnat et si oui si elle se classe dans les 3 dernières équipes.
* Si l’équipe de départ ne se retrouverait pas avec moins de 5 joueurs (en prenant compte de l’ensemble des transferts)

### Visualiser un match / le modifier

Renvoi vers la partie MatchManagement – voir ci-dessous

### Classement des équipes



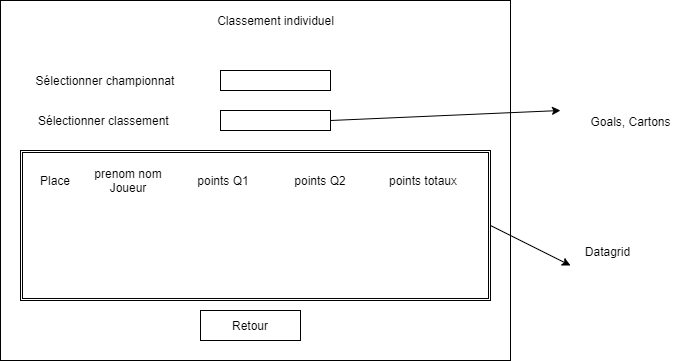
Affiche le classement des équipes par points, goals ou cartons pour un championnat sélectionné dans la liste par l’utilisateur

Est seulement affiché les équipes qui sont inscrites dans le championnat sélectionné.

Il est à noter que les cartons rouges valent 3 cartons jaunes pour le classement.

Il ne s’agit que d’un tableau en lecture.

### Classement des joueurs



Affiche le classement des joueurs par goals ou cartons pour un championnat sélectionné dans la liste par l’utilisateur.

Est seulement affiché la liste des joueurs inscrits dans une équipe qui est dans le championnat.

Il est à noter que les cartons rouges valent 3 cartons jaunes pour le classement.

Il ne s’agit que d’un tableau en lecture.

## MatchManagement

Un MDI Form contient toutes les autres formes de MatchManagement

Nous décrirons ci-dessous l’entièreté des mockups, un par un.

### Accueil



En sélectionnant le championnat, le datagridview affiche tous les matchs du championnat.

Ne peuvent être modifiés au niveau de la feuille de match que les matchs qui n’ont pas encore été joués (pour lesquels une carte ou un goal n’a pas encore été inscrit)

La feuille de résultat peut toujours être modifiée mais uniquement au niveau des goals si le match est considéré comme joué.

### Feuilledematch

Une image contenant capture d’écran

Description générée automatiquement

Seuls les joueurs faisant partie de l’équipe sont affichés dans la liste des joueurs. Un checkbox permet de les sélectionner facilement pour le match.

Les joueurs suspendus à la suite d’un carton rouge sont affichés en rouge et ne peuvent pas être inscrits

Les joueurs suspendus à la suite d’un carton jaune sont affichés en jaunes et ne peuvent pas être inscrits

Un message, avant sauvegarde, demande confirmation si les quotas de joueurs ne sont pas atteints.

### Inscription des scores

Une image contenant capture d’écran

Description générée automatiquement

Le champ Joueur reprend le prénom et le nom du joueur qui a déjà été inscrit pour un goal ou carte.

Changer le joueur est une combobox ne comprenant que les joueurs inscrits sur la feuille de match et permet de modifier goal ou carton ou d’en rajouter un.

La case temps doit être comprise entre 0 et 120

Les cartes ne peuvent être remplies que la première fois où on sauvegarde un résultat. En cas de forfait, il n’est pas possible de remplir les champs goals et cartons.

Une vérification est faite avant l’enregistrement pour voir si un joueur n’a pas marqué après avoir reçu un carton rouge.

# Listes erreurs

2 types d’erreurs sont prévues pour cette solution.

D’une part, les technicalErrors qui sont soulevées par la base de données au travers des triggers et des procédures stockées.

D’autre part, les BusinessErrors qui sont soulevées dans les couche Business de la solution.

## TechnicalErrors – SQLExceptions

|  |  |  |
| --- | --- | --- |
| **Numéro** | **Message** | **État** |
| 50001 | Les années doivent être supérieures ou égales à 1900 | 1 |
| 50002 | Ce championnat existe déjà | 2 |
| 50003 | Ce championnat n’existe pas | 2 |
| 50004 | La date de début du championnat est trop tardive | 3 |
| 50005 | il existe déjà une intersaison pour ce championnat | 2 |
| 50006 | il existe déjà un quarter pour ce championnat | 2 |
| 50007 | Il ne peut avoir que 2 quarters par championnat | 3 |
| 50008 | Cette équipe n’existe pas | 2 |
| 50009 | L'équipe est déjà inscrite à ce championnat | 3 |
| 50010 | La date de match ne tombe pas pendant un quarter existant | 4 |
| 50011 | les 2 équipes inscrites sont les mêmes | 4 |
| 50012 | un match similaire a déjà été prévu pour ce championnat | 4 |
| 50013 | Une mise à jour plus récente a été effectuée | 5 |
| 50014 | Un carton rouge a déjà été donné à ce joueur pendant ce match ! | 4 |
| 50015 | Il ne reste plus de suspension à cette carte rouge | 4 |
| 50016 | Ce carton rouge n’existe pas | 2 |
| 50017 | Un carton est obtenu entre 0 et 120 minutes de match | 6 |
| 50018 | Un goal est marqué entre 0 et 120 minutes de match | 6 |
| 50019 | Ce carton jaune n’existe pas | 2 |
| 50020 | Ce goal n’existe pas | 2 |
| 50021 | Une feuille de match existe déjà pour cette équipe et pour ce match ! | 4 |
| 50022 | Il existe déjà 2 feuilles de match pour ce match ! | 4 |
| 50023 | Cette feuille de match n’existe pas | 2 |
| 50024 | Il y a déjà 7 joueurs inscrits sur cette feuille de match | 4 |
| 50025 | Ce joueur n’est pas inscrit dans l’équipe | 4 |
| 50026 | Joueur inscrit avec un carton rouge, ne peut pas être inscrit | 4 |
| 50027 | Joueur inscrit avec autant ou plus de cartons jaunes que de matchs restants | 4 |
| 50028 | Joueur déjà inscrit sur la feuille de match | 2 |
| 50029 | L’équipe doit avoir entre 5 et 10 joueurs | 4 |
| 50030 | L’équipe compte déjà 10 joueurs | 4 |
| 50031 | un joueur ne peut pas être transférer lors d''un quarter | 4 |
| 50032 | un joueur ne peut être transféré que dans une des 3 dernières équipes du classement lors de l’intersaison | 4 |
| 50033 | Il ne peut avoir qu’une intersaison par championnat | 3 |
| 50034 | Un nom ne peut être utilisé que par une équipe | 3 |
| 50035 | Une équipe doit respecter le nombre de joueurs min et max pour s’inscrire dans un championnat | 4 |
| 50036 | Un joueur doit quitter son équipe avant d''en rejoindre une nouvelle | 2 |
| 50037 | Une des 2 feuilles d’équipe pour ce match n" pas remplie | 2 |
| 50038 | les années des 2 dates doivent être identiques | 2 |
| 50039 | L’année de date de début doit être la même année que son championnat | 2 |

## BusinessErrors

Les BusinessErrors sont reprises dans les classes suivantes :

* ChampionnatsService
* JoueursService
* TransfertsService
* ClassementEquipe

# Liste des règles et contrôles de la solution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Numéro** | **Règle** | **BDD** | **Procédure stockée** | **Trigger** | **Business - Form** |
| R1 | Un championnat possède une année | Contrainte not null |  |  | Générer un championnat |
| R2 | Il ne peut y avoir qu'un championnat par année | Contrainte unique | BackEnd.Championnat\_Add |  | Générer un championnat |
| R3 | L'année d'un championnat est supérieur ou égal à 1900 |  |  | FormatAnneesChampionnats | Générer un championnat |
| R4 | Il ne peut y avoir qu'une intersaison par championnat | Contrainte unique | BackEnd.CreationDivisionChampionnat  BackEnd.Intersaisons.Add | Intersaisons\_UniciteChampionnat | Générer un championnat |
| R5 | Pour chaque intersaison, l'année de date de début, de fin et du championnat doivent être identiques |  | BackEnd.CreationDivisionChampionnat  BackEnd.Intersaisons.Add |  | Générer un championnat |
| R6 | Il ne peut y avoir que 2 quarters maximum par championnat |  | BackEnd.CreationDivisionChampionnat  BackEnd.Quarters.Add | MaxQuarters | Générer un championnat |
| R7 | Pour chaque quarter, l'année de date de début, de fin et du championnat doivent être identiques |  | BackEnd.CreationDivisionChampionnat  BackEnd.Quarters.Add |  | Générer un championnat |
| R8 | le nom d'une équipe doit être unique | Contrainte unique | Equipes.UniciteNom |  |  |
| R9 | logoPath si null est généré automatiquement en noLogo | Valeur par défaut noLogo |  |  |  |
| R10 | Une équipe ne peut participer qu'une fois à un championnat | Contrainte unique equipeId / championnatId | BackEnd.EquipeParticipation\_Add |  | Générer un championnat |
| R11 | Une équipe doit avoir assez de joueurs pour s'inscrire à un championnat |  | BackEnd.EquipeParticipation\_Add | EquipesParticipation\_EquipeActive | Générer un championnat |
| R12 | Une équipe ne peut pas avoir des transfert dans l'année du championnat avant son inscription au championnat |  |  |  | Générer un championnat |
| R13 | Un joueur possède un nom et un prénom | Contrainte not null |  |  |  |
| R14 | Les points de chaque équipe a un match sont initialisés à 0 | Valeur par défaut 0 |  |  |  |
| R15 | Un booleen isPlayed décrit si un match a été joué (non par défaut) | Valeur par défaut 0 |  |  |  |
| R16 | Un match possède une date qui doit être comprise dans un quarter |  | BackEnd.Matchs\_Add BackEnd et MatchManagement.Matchs\_Update | Matchs\_WithinChampionnat | Calendrier des matchs |
| R17 | Un match ne doit pas avoir 2x la même équipe |  | BackEnd.Matchs\_Add BackEnd et MatchManagement.Matchs\_Update | Matchs\_SameTeams | Calendrier des matchs |
| R18 | Un match avec la combinaison équipe1 - équipe2 doit être unique dans un championnat |  | BackEnd.Matchs\_Add BackEnd et MatchManagement.Matchs\_Update | Matchs\_Doublons | Calendrier des matchs |
| R19 | Les matchs à domicile et à l'extérieur d'une équipe lors d'un championnat sont répartis équitablement entre les 2 quarters |  |  |  | Calendrier des matchs |
| R20 | Les matchs se jouent par défaut le week end, une équipe ne peut jouer qu'une fois par week end. En cas de trop de match par rapport au dates disponibles, l'utilisateur choisit les dates |  |  |  | Calendrier des matchs |
| R21 | Une équipe ne pas jouer 2 matchs le même jour ou à un jour d'interval |  |  |  | Calendrier des matchs |
| R22 | Un match du premier quarter, ne peut être assigné par l'utilisateur que dans le 1er quarter |  |  |  | Calendrier des matchs |
| R23 | Toute modification d'un match doit s'assurer que le match n'a pas déjà reçu un modification plus récente |  | BackEnd.Matchs\_Add BackEnd et MatchManagement.Matchs\_Update | Matchs\_UpdatePlusVieux | Calendrier des matchs |
| R24 | Un match est joué dès qu'un carton jaune est encodé et les points sont initialisés |  |  | CartonsJaunesHistory\_PlayedMatch | Inscriptions des résultats |
| R25 | Un match est joué dès qu'un carton rouge est encodé et les points sont initialisés |  |  | CartonsRougesHistory\_PlayedMatch | Inscriptions des résultats |
| R26 | Un match est joué dès qu'un goal est inscrit et les points sont comptés |  |  | GoalsHistory\_PlayedMatch | Inscriptions des résultats |
| R27 | Si une des 2 feuilles de match compte moins de 5 joueurs alors l’autre équipe gagne automatiquement |  |  | JoueursParticipationHistory\_CheckFeuillesDeMatch | Feuille de match |
| R28 | Si les 2 équipes ont moins de 5 joueurs sur leur feuille de match, celle avec le plus de joueur gagne le match |  |  | JoueursParticipationHistory\_CheckFeuillesDeMatch | Feuille de match |
| R29 | Les cartons rouges ne peuvent être reçus qu'entre la minute 0 et 120 d'un match |  |  | CartonsRougesHistory\_LimiteMinuteRecue | Inscriptions des résultats |
| R30 | A l'attribution d'un carton rouge, un joueur est suspendu pour 3 matchs | Valeur par défaut 3 |  | CartonsRougesHistory\_LimiteMinuteRecue | Feuille de match |
| R31 | Une carte rouge ne peut être attribuée qu'à un joueur qui joue le match |  |  | CartonsRougesHistory\_EstDansFeuilleDeMatch | Inscriptions des résultats |
| R32 | Pour qu'une carte rouge soit donnée, il faut que les 2 feuilles de matchs soient complétées et valides |  |  | CartonsRougesHistory\_EnoughJoueurs | Inscriptions des résultats |
| R33 | Toute modification d'un carton rouge doit s'assurer que la carte n'ait pas déjà reçu un modification plus récente |  |  | CartonsRougesHistory\_UpdatePlusVieux | Inscriptions des résultats |
| R34 | Un joueur ne peut recevoir qu'un carton rouge par match |  | BackEnd et MatchManagement.CartonsRougesHistory\_Add BackEnd et MatchManagement.CartonsRouges\_Update | CartonsRougesHistory\_MaxCartonParMatch | Inscriptions des résultats |
| R35 | Chaque match non joué par un joueur diminue la suspension d'un sur une carte rouge "active" |  | BackEnd et MatchManagement.CartonsRouges\_DiminueSuspension |  |  |
| R36 | Les cartons jaunes ne peuvent être reçus qu'entre la minute 0 et 120 d'un match |  |  | CartonsJaunesHistory\_LimiteMinuteRecue | Inscriptions des résultats |
| R37 | Toute modification d'un carton jaune doit s'assurer que la carte n'ait pas déjà reçu un modification plus récente |  |  | CartonsJaunesHistory\_UpdatePlusVieux | Inscriptions des résultats |
| R38 | Une carte jaune ne peut être attribuée qu'à un joueur qui joue le match |  |  | CartonsJaunesHistory\_EstDansFeuilleDeMatch | Inscriptions des résultats |
| R39 | Pour qu'une carte jaune soit donnée, il faut que les 2 feuilles de matchs soient complétées et valides |  |  | CartonsJaunesHistory\_EnoughJoueurs | Inscriptions des résultats |
| R40 | A l'attribution d'un carton jaune, il est actif | Valeur par défaut isActive 1 |  |  | Inscriptions des résultats |
| R41 | Les goals ne peuvent être inscrits qu'entre la minute 0 et 120 d'un match |  |  | GoalsHistory\_LimiteMinuteRecue | Inscriptions des résultats |
| R42 | Toute modification d'un goal doit s'assurer que le goal n'ait pas déjà reçu un modification plus récente |  |  | GoalsHistory\_UpdatePlusVieux | Inscriptions des résultats |
| R43 | Un goal ne peut être marqué que part un joueur qui joue le match |  |  | GoalsHistory\_EstDansFeuilleDeMatch | Inscriptions des résultats |
| R44 | Pour qu'un goal soit inscrit, il faut que les 2 feuilles de matchs soient complétées et valides |  |  | GoalsHistory\_EnoughJoueurs | Inscriptions des résultats |
| R45 | Il ne peut y avoir qu’une feuille de match pour une équipe pendant un match précis. |  | BackEnd et MatchManagement.FeuilleDeMatch\_Add BackEnd et MatchManagement.FeuilleDeMatch\_Update | FeuillesDeMatch\_MaxFeuilleParEquipe | Feuille de match |
| R46 | Il ne peut avoir que 2 feuilles de match pour un même match. |  | BackEnd et MatchManagement.FeuilleDeMatch\_Add BackEnd et MatchManagement.FeuilleDeMatch\_Update | FeuillesDeMatch\_MaxFeuilleParMatch | Feuille de match |
| R47 | Toute modification d'une feuille de match doit s'assurer que la feuille n'ait pas déjà reçu un modification plus récente |  | BackEnd et MatchManagement.FeuilleDeMatch\_Update | FeuillesDeMatch\_UpdatePlusVieux | Feuille de match |
| R48 | Une équipe ne peut être composée que de 10 joueurs maximum |  |  | TransfertsHistory\_NombreJoueursMax TransfertsHistory\_NombreJoueurs | Transférer des joueurs |
| R49 | Un joueur ne peut pas changer d’équipe lors d’un quarter. |  |  | TransfertsHistory\_DateTransfert | Transférer des joueurs |
| R50 | Un joueur ne peut changer d’équipe à l’intersaison que si son équipe possède plus de 5 joueurs et que l’équipe dans laquelle il va est classée parmi les 3 dernières du championnat. |  |  | TransfertsHistory\_DateTransfert | Transférer des joueurs |
| R51 | Un joueur ne peut pas jouer dans 2 équipes en même temps |  | BackEnd.Tansferts\_Add | TransfertsHistory\_DeuxEquipes | Transférer des joueurs |
| R52 | Toute modification d'un transfert doit s'assurer que le transfert n'ait pas déjà reçu un modification plus récente |  | BackEnd et MatchManagement.JoueursParticipation\_Add | TransfertsHistory\_UpdatePlusVieux | Transférer des joueurs |
| R53 | Il ne peut avoir que 7 joueurs maximum par feuille de match |  | BackEnd et MatchManagement.JoueursParticipation\_Add | JoueursParticipationHistory\_MaxJoueursParFeuille | Feuille de match |
| R54 | Un joueur ne peut être inscrit que s’il joue dans l’équipe de la feuille de match |  | BackEnd et MatchManagement.JoueursParticipation\_Add | JoueursParticipationHistory\_EstDansEquipe | Feuille de match |
| R55 | Un joueur qui a une carte rouge active ne peut pas être inscrit sur la feuille de match |  | BackEnd et MatchManagement.JoueursParticipation\_Add | JoueursParticipationHistory\_CartonRouge | Feuille de match |
| R56 | Un joueur qui à le même nombre ou plus de carte jaune que de match restant à jouer pour son équipe, il ne peut pas être inscrit sur la feuille de match. |  | BackEnd et MatchManagement.JoueursParticipation\_Add | JoueursParticipationHistory\_CartonJaune | Feuille de match |

# Lessons learned

Autant des erreurs ont été réalisées dans la gestion de ce projet et dans l’approche de ce problème, autant l’identification de ces erreurs et leur résolution m’a permis d’apprendre beaucoup de chose sur le travail d’analyse et de développement mais également sur mon approche d’un problème.

### Ne rien faire à l’avance

Je pense que le premier point que ce travail m’a appris et il s’agit sans doute d’un des plus important est de ne rien faire à l’avance. J’ai travaillé par couche, ce qui n’est pas forcément mauvais mais m’a demandé beaucoup de travail de réécriture et d’adaptation sur des procédures stockées et triggers. De plus, certaines de ces procédures se sont révélées par la suite inutiles.

Il aurait été, je pense, plus efficace de faire un socle de base à chaque couche et de développer par la suite selon les besoins. Les mockups (réalisés avant la base de données) de limiter au maximum ce type d’erreurs et sont vraiment important à la réalisation d’un projet de ce type.

### Bien identifier la technologie nécessaire

Au départ, la partie BackEnd a été développée en utilisant Entity Framework – CodeFirst. Je pensais que cette solution me permettrait plus de flexibilité au niveau du code. Malheureusement, je n’avais pas identifié le problème des procédures stockées, CodeFirst limitant le nombre de procédures stockées mappées par entité et ce travail nous interdisant tout accès direct aux tables cela a transformé une flexibilité en quelque chose de très lourd.

Mais cette mauvaise identification m’a permis d’apprendre plus en profondeur le code first et la DB first, leurs différences techniques et d’approches. De plus, en Code First, l’utilisation de LINQ m’a permis d’améliorer grandement mon utilisation des prédicats.

### Découper son code et tester sur des petits morceaux de code

Il s’agit du premier « gros » projet de développement à réaliser pour l’Ephec, plein d’enthousiasme et ayant pour une fois des journées entières à consacrer au codage, je me suis laissé aller à développer de gros morceaux de code et de grosses fonctionnalités pour ne faire mes tests qu’à la fin de la journée et avec peu de commentaires.

Il est inutile de dire que très rapidement, je me suis rendu compte de l’importance de diviser au maximum les fonctions en méthodes, de commenter mon code et de rendre lisible et facilement compréhensibles mes variables et méthodes.

### DATAGRIDVIEW – DATATABLE - Dataview

L’utilisation des forms pour ce projet m’a très vite imposé l’utilisation de nombreux DataGridView pour rendre l’interface le plus amical possible avec l’utilisateur. Matière survolée aux cours.

Ces utilisations m’ont demandé beaucoup de recherches sur la nature même des DataGridView et leur approche. A travers ses recherches j’en ai également beaucoup appris sur les DataTable et DataView.

### Le mieux est l’ennemi du bien

Encore maintenant alors que tout fonctionne bien, en regardant mon code le plus ancien, j’ai l’envie de le refactoriser, de le simplifier, de l’améliorer.

Je me rend compte qu’à travers ce projet mon code s’est amélioré en de nombreux aspects. Malgré tout, il me faut laisser certaines parties de code « anciennes ». Ces parties ont fait leurs preuves à travers de nombreux tests et fonctionnent pour ce qu’on leur demande.

Certaines modifications réalisées uniquement pour améliorer le code m’ont amené à bien plus de changements que je ne le pensais et ont été énergivores. Ce n’est pas forcément une mauvaise chose car cela m’a appris d’autres choses mais dans un projet il faut également savoir gérer son temps.

# Conclusion

Un tel projet n’est jamais évident. Il y a eu des moments de doutes, de questionnements, des retours en arrières dans une approche, un client n’est jamais précis dans sa demande, son contexte n’est jamais bien défini.

J’espère que la réflexion au travers des différents chapitre a pu mettre en lumière de manière suffisamment simple et compréhensible le résultat de ces doutes.

Les différentes étapes : schéma entité-association – database – mockups – code m’ont permis à chaque fois d’affiner cette pensée et de préciser ma solution.

Je pense offrir ici une solution qui répond à toutes les demandes critiques et propose même certaines bases permettant une modification future plus facile.

Ce projet m’a énormément appris et m’a ouvert l’appétit pour faire de plus grands projets encore, j’ai hâte de travailler en équipe sur un grand projet.

Je finirai en remerciant Monsieur Fievez pour m’avoir donné cette opportunité de développer ma pensée et mes connaissances.

1. Ce travail étant de nature académique, il a été décidé de laisser quelques tables sans identifiant GUID pour démontrer la capacité à utiliser les 2 techniques. [↑](#footnote-ref-1)
2. Ce travail étant de nature académique, il a été décidé de laisser quelques tables sans identifiant GUID pour démontrer la capacité à utiliser les 2 techniques. [↑](#footnote-ref-2)
3. Ce travail étant de nature académique, il a été décidé de laisser quelques tables sans identifiant GUID pour démontrer la capacité à utiliser les 2 techniques. [↑](#footnote-ref-3)