# Traduction de l’énoncé du projet

## 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, celles-ci diminuent de 1
* 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

Une image contenant texte, carte

Description générée automatiquement

# Schéma Relationnel

Une image contenant capture d’écran

Description générée automatiquement

# Avant-Propos

Certaines procédures testent des conditions qui sont déjà vérifiées de part les contraintes de la base de données. Cette étape à pour but de fournir des messages d’erreurs personnalisés.

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.

# Tables SQL

## 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 à la création de tout nouveau championnat.

Aucun de ses champs ne peut être nul.

Année est un champ 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 (int @année) : 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.

## 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 (datetime @DateDebutChampionnat, uniqueidentifier @ChampionatId) : 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.

## 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

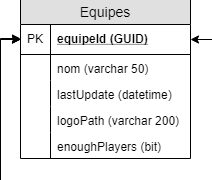
### Triggers :

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

### Procédures stockées :

* BackEnd.CreationDivisionChampionnat (datetime @DateDebutChampionnat, uniqueidentifier @ChampionatId) : 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.

## 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.

enoughPlayers est utilisé pour savoir si l’équipe a assez de joueurs pour s’inscrire dans un championnat (entre 5 et 10 joueurs). Par défaut la valeur est fausse (0).

### Triggers :

* Equipes\_UniciteNom : vérifie que le nom d’équipe n’est pas déjà utilisé.
* TransfertHistory\_ActiveTeam : sur la table TransfertHistory, active une équipe si elle a entre 5 et 10 joueurs inscrits

## 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 participation est unique par 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 au championnat avant de l’inscrire.

### Procédures stockées :

* BackEnd.EquipeParticipation\_Add (uniqueidentifier @equipeId, uniqueidentifier @championnatId) : vérifie que le championnat et l’équipe existe et si la combinaison n’existe pas déjà et sinon crée l’association de participation.

## 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

## Table Matchs

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, 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.

### 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.

## Table CartonsRougesHistory

Une image contenant capture d’écran

Description générée automatiquement

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 2 clés étrangères : joueurId et matchId. NombreSuspensionRestante et minuteRecu sont des int.

Aucun de ses champs ne peut être nul.

minuteRecu doit être inclus entre 0 et 120 (sans prolongations un match dure 90 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.

## Table CartonsJaunesHistory

Une image contenant capture d’écran

Description générée automatiquement

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 2 clés étrangères : joueurId et matchId. minuteRecu est un int et isActive est un bit initialisé par défaut à 1.

Aucun de ses champs ne peut être nul.

minuteRecu doit être inclus entre 0 et 120 (sans prolongations un match dure 90 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.

## Table GoalsHistory

Une image contenant capture d’écran, texte

Description générée automatiquement

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 2 clés étrangères : joueurId et matchId. minuteMarque est un int.

Aucun de ses champs ne peut être nul.

minuteMarqué doit être inclus entre 0 et 120 (sans prolongations un match dure 90 minutes)

lastUpdate est là pour faire un suivi mais également s’assurer que lors de toute modification d’un goal 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.

## 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 : joueurId 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.

## 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.

## 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

# Listes erreurs

|  |  |  |
| --- | --- | --- |
| **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 |

# Schémas

MatchManagement

BackEnd

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)