Système de Gestion des Bases de Données

Deuxième année Informatique

Rapport de projet

Base de données modélisant un jeu de cartes - Carteirb

Développeurs :
Johan Chataigner
Emeric Duchemin
Laurent Genty
Dylan Hertay

Responsable :
Sylvain Lombardy

11 décembre 2019

Table des matières

1 Introduction

Notre projet cherche à implémenter une base de données représentant un jeu de cartes. Nous avons choisi dans le cadre de notre exercice, le jeu de cartes à jouer Yu-Gi-Oh. Notre choix s'est tourner vers ce jeu de cartes car il est complet en terme de caractéristiques et d'attributs pour chaque carte (attaque, défense, description) et pour ses éditions (chaque carte a plusieurs éditions). C'est aussi parce que nous connaissons ce jeu que nous l'avons sélectionné. Afin de la rendre facilement manipulable, par le plus grand nombre d'utilisateurs, nous avons aussi choisi d'implémenter une interface web. Toujours dans l'optique de rendre accessible les tables, nous avons essayé de cacher le plus possible la réalisation de la base de données de sorte que l'utilisateur puisse remplir, au travers de champs, sa base de données de cartes. Voici donc le rapport explicitant la structure de notre base de données, ainsi que les différents cas d'utilisation que pourra envisager un futur utilisateur.

2 Modélisation des données

Nous avons fait le choix dans notre base de données de définir des parties qui seraient relatives à des tournois. C'est-à-dire qu'une partie a forcément lieu durant un tournoi. Le champ id_tournoi d'une partie n'est donc jamais null. Ce choix s'explique par le fait que mettre des rencontres hors tournoi officiel pourrait amener à des dérives (triches, écart de rencontres bien trop élevé entre joueur).

Nous avons aussi réalisé une table des caractéristiques. En effet, cette table nous permet de gagner en complexité spatiale de notre table et permet de plus, une plus grande évolutivité de la base de donnée. On peut grâce à cette table fournir un nombre de caractéristiques aussi grand que l'on désire. Contrairement à la réalisation naïve de la base (mettre un champ attaque, un champ défense), avec la table des caractéristiques les champs ne sont pas nommés. Ainsi peu importe le type de cartes on aura donc pas forcément des attaques ou des défense mais peut être des caractéristiques, mana par exemple.

2.1 Remarques et spécificités du modèle de données

Nous avons choisi de relier Joueurs et Parties. Chaque partie est composée d'un deck à partir duquel on peut retrouver le joueur possédant le deck. La liaison entre les entités Joueurs et Parties permet de stocker l'adversaire du joueur avec ce deck. Il est donc inutile de stocker l'id des deux decks. Par contre, il y aura deux fois plus de lignes dans la table Parties puisqu'il faut refaire la même chose pour l'adversaire.

Cependant, en termes d'utilisation, si pour une partie, on veut connaitre les adversaires d'un joueur, il suffit d'aller chercher l'id_joueur dans la table Joueurs pour obtenir ses informations. lorsque l'on recherche une partie et que l'on veut connaître seulement les adversaires, il est plus facile de relier l'adversaire, on évite les jointures entre les tables (Decks,Parties) et (Decks,Joueurs).

Nous avons aussi autorisé une partie à avoir plusieurs decks. En effet dans une partie de tournoi, généralement, celle-ci se déroule sur le principe de plusieurs matchs (exemple j'ai gagné deux matchs, adversaire n'en a gagné qu'un, je remporte donc la partie). De fait, lors d'une partie un joueur va potentiellement vouloir changer de deck d'un match à un autre (afin de surprendre son adversaire). La modélisation de notre base de données le permet.

2.2 Modèle entité association

Durant la phase de conception d'un projet de gestion de base de données, il y a un écart entre la conception et la réalisation. En effet, durant la phase de conception, nous souhaitons éviter toute redondance, nous normalisons le plus possible et nous faisons en sorte que notre modèle soit le plus optimisé possible. En revanche il est possible que le modèle implémenté soit différent du premier conçu et ce à cause d'oublis ou bien de contraintes n'ayant pas fait surface au départ.

Nous allons donc voir les potentielles différences dans les deux modèles.

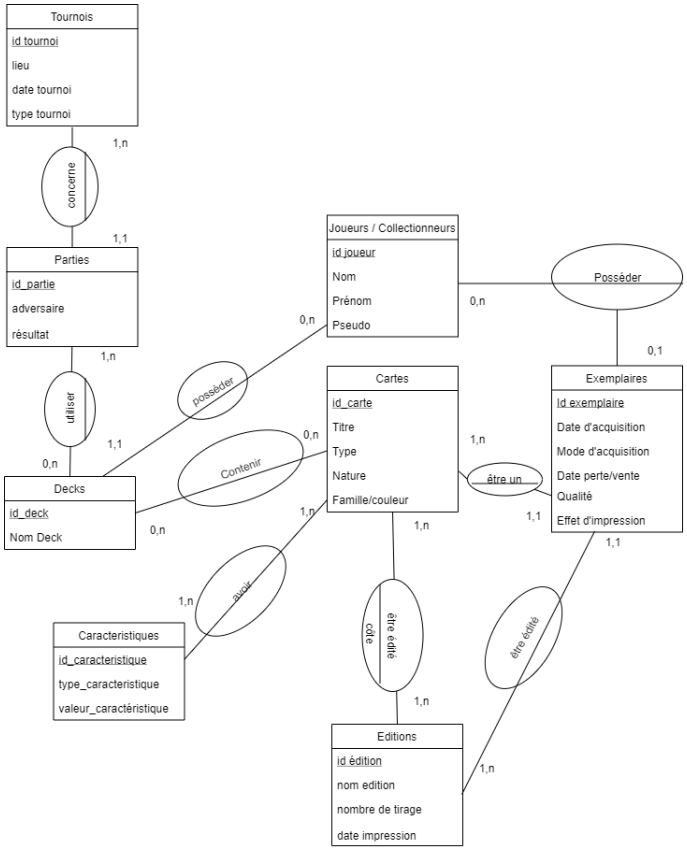

FIGURE 1 – Modèle conceptuel de données avant réalisation du projet

Comme nous pouvons le voir, nous avons un peu modifié le modèle conceptuel. En effet, malgré un effort de notre part en amont afin de rendre notre modèle plus évolutif possible et le plus versatile, il s'avère que certaines informations était erronées.

Nous avons enlevé l'attribut `adversaire` d'une partie dans la mesure où pour qu'une partie existe, cela doit être contre un joueur existant. Nous aurions pu faire le choix de garder cet attribut cependant nous l'avons représenté en tant qu'association entre les deux entités dans la mesure où l'adversaire doit exister.

On pourrait croire qu'il y a un cycle dans notre MCD : joueur est relié à parties, qui est reliée à deck nous permettant de retrouver son créateur et donc le joueur. Cependant ces liens n'ont pas le même sens. En effet, une partie va donc être unique par joueur et donc va représentée par une ligne dans cette table. On pourra retrouver le joueur ayant fait cette partie grâce au deck avec lequel il a joué durant cette partie contre l'adversaire qui lui est représenté par l'association.

Après réflexion, nous pensons que la manière avec laquelle nous avons modélisé le MCD n'est pas bonne. Malgré le fait qu'il ne s'agisse pas de redondance entre l'id du joueur qui a joué le deck pendant la partie et l'adversaire, il s'avère que pour une vraie partie (donc 2 joueurs qui s'affrontent) nous sommes obligés de créer 2 parties : une pour chaque joueur (et donc pour chacune stockant le résultat du match et l'opposant).

De plus, nous avons aussi rajouté un attribut image à notre entité Cartes. C'est un petit changement qui reste cependant discutable quant au fait qu'une carte puisse avoir une image différente selon les éditions :

Concernant les parties, nous avons longtemps discuté sur comment nous allons gérer les informations. En effet,

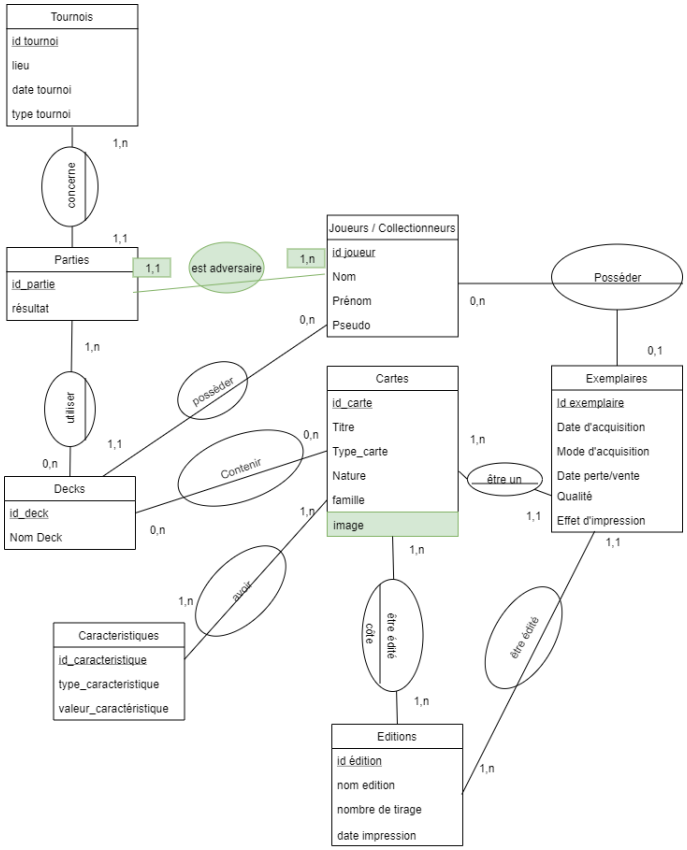

FIGURE 2 – Modèle conceptuel de données après réalisation du projet

une partie se compose de plusieurs joueur, ayant un résultat de partie pour chacun et chaque joueurs peuvent utiliser plusieurs decks dans la partie.

Nous aurions pu partie sur cette implémentation :

Comme nous pouvons le voir sur la Figure 4, nous aurions pu faire une association ternaire qui regroupe donc n joueurs (2 en l'occurrence) avec plusieurs decks dans plusieurs parties. Cette modélisation aura pour effet de créer une table d'association Jeu ayant toutes les informations à l'intérieur.

Cependant, nous avons fait le choix de ne pas partir sur cette implémentation dans la mesure lorsque nous allons créer le modèle relationnel, nous allons voir qu'il y a de la redondance et des problèmes associés.

En effet dans la table d'association nous aurons pour une ligne :

- id_joueur (j1 ou j2 ?)
- resultatJ1
- resultatJ2
- id_deck (j1 ou j2)

Cependant, comme nous pouvons le voir, nous aurons des incohérence concernant les données : en effet l'id du deck présent dans la table correspond-il au deck du joueur qui est présent dans la ligne de données, ou bien l'adversaire (grâce à l'association entre decks et joueurs) ? De plus, quoiqu'il arrive, nous devrons donc avoir une autre ligne pour le Jeu de l'autre joueur j2 et donc il y aura une redondance concernant les informations du résultat.

Enfin, utiliser une association ternaire n'est pas une bonne habitude, c'est donc pourquoi nous sommes partis sur cette modélisation qui admet une redondance (création de 2 parties pour une seule vraie partie) mais qui

FIGURE 3 – Exemple d'artworks différents selon les éditions

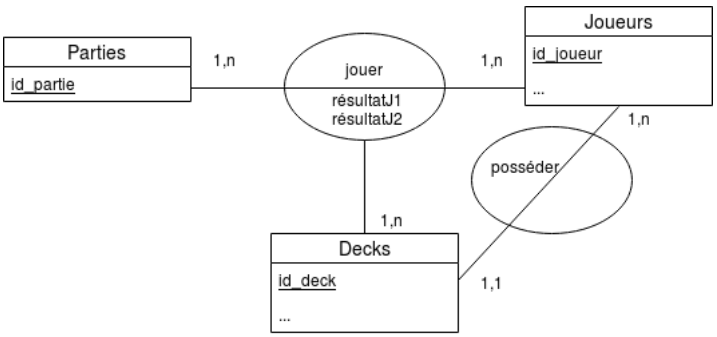

FIGURE 4 – Potentielle solution pour les parties

nous permet de simplifier les requêtes futures.

Un autre point sur lequel nous avons fait des choix est celui concernant les caractéristiques. En effet, nous avons considéré qu'une carte pouvait avoir plusieurs caractéristiques selon les éditions et selon les règles évolutives (cartes bannies ou nerfées).

Sur la Figure 5 nous pouvons voir les différentes caractéristiques qui peuvent être ajoutées ou bien modifiées sur une carte.

Nous étions voulions au départ mettre ces informations dans l'association entre cartes et éditions cependant d'un point de vue de la normalisation des associations, ce n'était pas bon.

Après développement, notre choix est largement discutable.

2.3 Différentes opérations supportées par la base de données

La base de données supporte, au vu de ce diagramme entité association, différents types d'opérations. Les différentes consultations classiques, à savoir :

1. Affichage des cartes, des exemplaires, des joueurs...

FIGURE 5 – Caracteristiques pouvant être changées

2. Affichage des cartes par types, affichage des cartes selon un certain type.
3. Affichage de joueur n'ayant pas une caractéristiques, ou de decks n'ayant pas une caractéristique.
4. Affichage des différentes confrontations et vainqueurs.
5. Affichage des pourcentage de victoire des joueurs.
6. Affichage du pourcentage d'utilisation dans les decks des différentes cartes.

On peut aussi ajouter à la base de données classiquement, des joueurs, des exemplaires, des cartes, des tournois, mais aussi des parties relatives à un tournoi, des caractéristiques relatives à une carte. On pourra aussi à contrario, retirer de la base des cartes (on retirerait alors les exemplaires de cette cartes qui n'aurait plus lieu d'exister), des tournois (auquel cas les parties qui lui sont associées sont elles aussi retirées puisqu'elles n'auraient alors plus de sens), et tous les autres objets ajoutables.

3 Passage au schéma relationnel

3.1 Schéma relationnel

La figure ci-dessus représente la structure finale de notre base de données, créée via le script SQL `backend/create_database.sql`. Dans le but d'implémenter les requêtes demandées dans le sujet ainsi que d'autres de notre choix, nous avons effectué quelques modifications sur notre modèle initial :

- Pour une question d'intégrité, la relation `Parties` contenant le champ `adversaire` est maintenant reliée via celui-ci à la relation `Joueurs`. `adversaire` est dorénavant une clé étrangère référençant `Joueurs`. L'ajout de cette contrainte permet de s'assurer que l'adversaire est bien un joueur stocké dans la base et on peut pour une partie donnée, accéder aux deux participants. Cela a notamment servi pour la requête qui, pour chaque joueur, affiche le joueur qu'il a le plus vaincu ;
- Pour mesurer la rareté d'une carte, nous avons complété la relation `Editions` avec un attribut `date_impression`, ainsi on mesure bien la valeur d'une carte avec une composante temporelle qui est ce nouvel attribut en plus de son nombre d'impressions ;
- Pour embellir la partie frontend et rendre la base plus complète, nous avons rajouté un champ `image` qui contient l'url de l'image associée à la carte en question. Ce champ peut être `NULL`. Il s'agit simplement d'un affichage supplémentaire.

Du côté des types choisis pour nos colonnes, plusieurs explications s'imposent. Dans un premier temps, nous avons choisi un `varchar(1024)` pour la colonne `valeur_caracteristique` car une caractéristique peut être l'effet d'une carte donc une description longue peut être requise. De même pour le champ `image` d'une carte, nous avons pris en compte qu'une URL peut être longue.

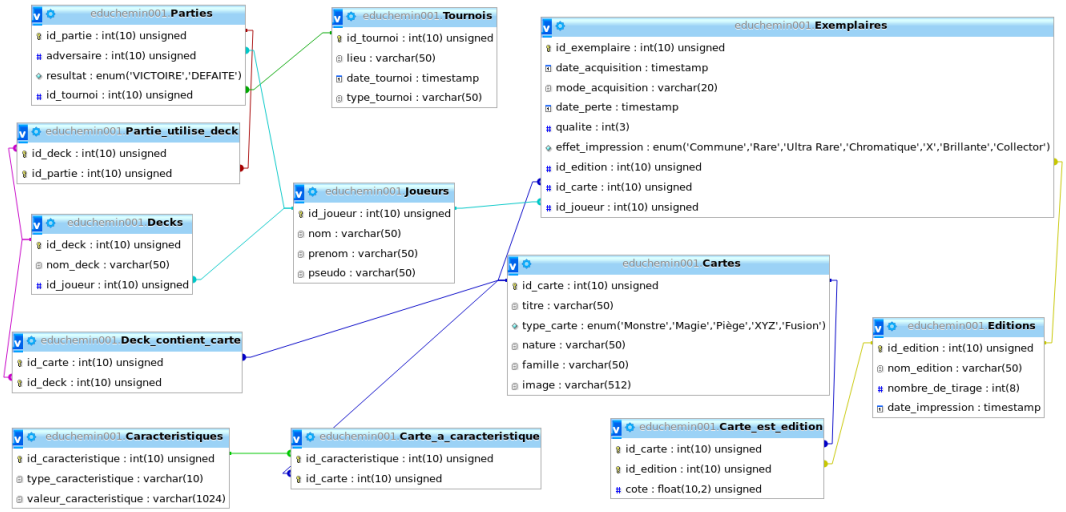

FIGURE 6 – Schéma relationnel obtenu via *phpMyAdmin* après création de la base

Enfin, nous avons, pour certains champs, opté pour des `ENUM`, pour limiter les valeurs de saisies possibles par les utilisateurs par exemple. En effet, si la valeur saisie ne correspond pas à une des valeurs énumérées l'insertion dans la base n'est pas effectuée. Par exemple pour `resultat` dans `Parties`, cela évite d'avoir "Victoire", "victoire", "VICTOIRE", etc... pour le même résultat d'un match. Ainsi on facilite surtout les requêtes. Par exemple, sélectionner uniquement les victoires nécessite seulement **WHERE** resulat = 'VICTOIRE'

au lieu de devoir tester toutes les orthographes et écritures possibles de "victoire". Cependant ceci implique que pour pouvoir insérer une nouvelle valeur qui n'est pas dans l'`ENUM`, il faille à chaque fois modifier celui-ci.

3.2 Contrainte d'intégrité et dépendance fonctionnelle

Nous avons plusieurs contraintes différentes. Concernant les contraintes de clés primaires, il s'agit tout simplement des attribut(s) noté(s) avec une petite clé à côté dans le modèle relationnel montré plus haut sur la Figure 6.

3.2.1 Contraintes FOREIGN KEY

- Exemplaires
  - Exemplaires.id_edition **référence** Editions.id_edition
  - Exemplaires.id_carte **référence** Cartes.id_carte
  - Exemplaires.id_joueur **référence** Joueurs.id_joueur
- Decks
  - Decks.id_joueur **référence** Joueurs.id_joueur
- Parties
  - Parties.id_tournois **référence** Tournois.id_tournois
  - Parties.adversaire **référence** Joueurs.id_joueur

- Cartes_a_caracteristiques
  - Cartes_a_caracteristiques.id_caracteristique **référence** Caracteristiques.id_caracteristique
  - Cartes_a_caracteristiques.id_carte **référence** Cartes.id_carte
- Cartes_est_edition
  - Cartes_est_edition.id_edition **référence** Editions.id_edition
  - Cartes_est_edition.id_carte **référence** Cartes.id_carte
- Deck_contient_carte
  - Deck_contient_carte.id_carte **référence** Cartes.id_carte
  - Deck_contient_carte.id_deck **référence** Decks.id_deck
- Parties_utilise_deck
  - Parties_utilise_deck.id_partie **référence** Parties.id_partie
  - Parties_utilise_deck.id_deck **référence** Decks.id_deck

Ces contraintes permettent donc d'assurer l'intégrité des informations que nous rentrons dans la base de données. Nous gérons ces contraintes dans le script de création de base de données dans lequel nous ajoutons les contraintes et nous vérifions que si l'on supprime une donnée qui a un champ dont on est dépendant autre part, les données sont bien supprimées avec toutes les dépendances.

3.2.2 Contraintes CHECK

Les contraintes CHECK permettent de vérifier la qualité des informations rentrées. En l'occurrence nous avons une contrainte concernant les dates d'acquisition et de pertes pour un exemplaire de carte. En effet, si la date de perte n'est pas nulle (ce qui est permis) nous ne pouvons pas faire en sorte que la date de perte soit inférieure à celle d'acquisition. La qualité quant à elle, est notée grâce à un chiffre compris entre 0 et 100, nous vérifions donc aussi cela dans la création des contraintes lors de la création de la base de données.

De plus, la côte qui est stocké dans la table `Carte_est_edition` doit forcément être supérieure à 1.

Concernant les éditions, le nombre de tirages doit être supérieur ou égal à 0.

3.2.3 Contraintes UNIQUE

Mise à part les identifiants qui doivent être uniques, nous avons ajouté des contraintes permettant d'assurer l'unicité de certains attributs comme notamment le titre des cartes. En effet, nous partons du principe qu'une carte ayant le même nom qu'une autre est forcément la même.

3.3 3ème forme normale

Nous avons essayer de respecter au maximum les règles des formes normales, afin d'éviter les redondances, la perte de données, limiter les incohérences (avec des données qui se contredisent) et améliorer le traitement des données. Pour commencer, nous respectons la première forme normale puisque chaque attribut est atomique. Chaque attribut n'a qu'une seule valeur et ne contient pas de listes. Il y a très peu de tables contenant plus d'une clé primaires et ces tables sont toutes sans attribut hormis Carte_est_edition qui contient la cote qui dépend de de id_carte et de id_edition. Tous les criètes sont regroupés pour respecter la deuxième forme normale. De plus, chaque clé candidate dépend uniquement de la clé primaire (ou groupement de clé qui représente la clé primaire) et ce pour chaque entité. Notre modèle respecte donc les 2 premières formes ainsi que cette règle, donc il vérifie la 3ème forme normale. On respecte aussi la forme normale de Boyce-Codd.

4 Création de base et requêtes proposées

Choix de la base et création des tables

Dans la création de nos tables nous avons fait le choix de conserver les auto increments natifs en mySQL. En effet lorsque nous voulions faire un id cette solution était la plus rapide à mettre en oeuvre et permettait d'assurer un résultat cohérent (chaque carte, par exemple, qui est ajoutée se voit affecter un id différent). Cependant nous reconnaissons que la meilleure façon de procéder aurait été de réaliser non pas un auto increment mais un i_uid. En effet c'est par exemple comme cela que fonctionne une édition d'une carte yu-gi-oh (CHIM-FRSE1 ici pour une carte chimère).

Dans notre création de base de données, nous avons créé des vues, des contraintes, mais pas de triggers par manque de temps ; Cependant, nous avons ajouté des pseudos-triggers concernant la suppression des lignes dans la mesure où dans notre script de création de base de données nos contraintes de clés étrangères sont implémentées avec `ON DELETE CASCADE` qui réalise la suppression des lignes où les clés apparaissent.

Nous avons, de plus, lancé depuis phpmyadmin notre script de création de table et notre script de peuplement afin de pouvoir dès le départ avoir une base de travail depuis le site web. Nous n'avons pas non plus permis de relancer le script de création de table depuis la partie utilisateur. En effet relancer ce script aurait effacé toutes les données de la table. Or ce que l'on souhaite avec une telle base de données, c'est que chaque personne ait accès sur son propre poste de travail à la même table des données. Ainsi pour réinitialiser toute la base, il faut avoir directement (avec phpmyadmin) accès à la base.

5 Choix des technologies

Concernant nos choix pour les technologies nous avons choisi de partir sur les suivantes :

- langage : SQL
- SGBD : MySQL
- frontend : php
- backend : php
- application web SGBD annexe : phpMyAdmin
- hébergement : session de l'école

Nous avons fait ces choix dans la mesure où il s'agit de notre première application de gestion de base de données. De ce fait, l'utilisation de MySQL comme SGBD était tout naturel. Facile d'utilisation, léger, user-friendly, peu verbeux nous permettait de facilement appréhender les bases de données.

Concernant le choix de développement front et back, nous avons souhaité le faire en php. Des ressources étaient déjà donnés afin de pouvoir démarrer et nous avons donc choisi de nous y atteler afin de pouvoir avoir une première expérience en Web PHP.

Nous avons aussi utilisé le SGBD en ligne phpMyAdmin. Ce dernier nous a permis de pouvoir créer en locale la base de données et de pouvoir effectuer nos tests avant de pousser les informations sur les bases de données personnelles à l'école. Nous pouvions donc tester nos scripts de backend afin de savoir si la base de données supportait bien les requêtes que nous lui demandions.

Enfin, concernant l'hébergement, nous avons choisi d'utiliser nos liens pédagogiques de l'école. Ils nous permettait de faire du PHP et de plus, nous n'avions pas besoin de payer quoique ce soit afin de pouvoir héberger ces données. Afin donc de pouvoir traiter notre connection à la base de données, nous avons utilisé la bibliothèque d'objets mysqli afin de pouvoir manipuler la BDD et effectuer des requêtes dessus.

Le seul point négatif étant que durant toute la durée du projet, nous n'avons pas réussi à trouver un moyen de pouvoir travailler depuis chez soi sur le site personnel, il fallait en effet se trouver sur un pc de l'école.

6 Application

Le site implémenté se nomme Carteirb, il va permettre de modifier la base de données, de consulter cette base et d'implémenter les requêtes spécifier par le sujet.

6.1 Notice d'utilisation et architecture globale

A l'ouverture du site, on arrive sur la page d'accueil du site. Il faudra alors se connecter avec un login ainsi qu'un password. Ces informations de connections dépendent des joueurs présents dans la base de données. Afin de pouvoir essayer, vous pouvez prendre le joueur suivant pour vous connecter : (nom,prénom) Seto,Kaiba.

L'utilisateur pourra ajouter les exemplaires qu'il possède, modifier ses decks (ajouter ou supprimer un deck et son contenu), consulter les decks de ces adversaires (pour adapter sa stratégie en fonction des decks joués) et consulter la liste des match pour chaque tournois.

Les onglets disponible sont :

- `/index.php` Accueil : mène vers la page d'Accueil.
- `/Cards.php` Cartes : lister les cartes, ajouter une carte, ajouter une carte à un deck, faire une recherche de carte et en supprimer une.
- `/Joueurs.php` Joueurs : lister les joueurs, en ajouter, supprimer des joueurs.
- `/Joueurs.id=XX.php` Mon profil : liste ses exemplaires, ajouter un exemplaire, en supprimer un, voir les cartes associées aux exemplaires
- `/Tournois.php` Tournoi : lister les tournois, ajouter un tournoi, supprimer un tournoi, modifier un tournoi, ajouter des parties à un tournoi
- `/Decks.php` Decks : lister les decks, ajouter un deck, ajouter un deck, supprimer un deck,
- `/Editions.php` Editions : lister les éditions, ajouter une édition, ajouter une carte à une édition, supprimer une édition
- `/Parties.php` Parties : lister les parties, ajouter une partie, ajotuer des decks à une partie
- `/Exemplaires.php` Exemplaires : lister ses exemplaires (du joueur connecté), supprimer un exemplaire, ajouter un exemplaire, aller voir la carte associée
- `/Contact.php` A propos de nous : infos supplémentaires concernant les développeurs du site.

6.2 Features/Requêtes demandées qui ont été réalisées

Nous avons effectué la liste des features demandées et implémentées sur le site, des pages lançant ces différentes requêtes :

- Liste des cartes d'un certain type : `CardsType.php`
- Liste des cartes qui ne font partie d'aucun deck : `CardsNoDeck.php`
- La liste des joueurs qui n'ont fait aucune partie (ce sont juste des collectionneurs) : `PlayerNoGames.php`
- La liste des joueurs, avec le nombre de cartes qu'ils possèdent : `Joueurs.php`
- La liste des joueurs classés par ordre décroissant selon la valeur de leur collection (la valeur d'une carte étant estimée au produit de sa côte par son état) : `PlayerCollectionValue.php`
- La liste des cartes avec le nombre de joueurs qui les utilisent dans leurs decks : `CardsUsedByPlayer.php`
- La liste des joueurs possédant le plus de cartes rares (date d'impression antérieure à 2000 ou tirage inférieur à 100) : `PlayerWithMostRares.php`
- + requêtes de création, suppression et visualisation de chaque table

A noter que ces pages sont disponibles à la navigation depuis la page `Cards.php` pour les requêtes concernant les cartes ou `Joueurs.php` pour les requêtes des joueurs (ou bien ajouter l'url à la main).

De plus, concernant le calcul du nombre de cartes rares, nous avons fait le choix de modifié la requête. En effet, comme vu précédemment dans notre MCD, nous avons stocké l'effet d'impression d'un exemplaire d'une carte. Ce dernier peut donc être égal à une carte commune, rare, ultra rare, collector... Nous avons donc fait le choix de compter le nombre de cartes selon si elles sont d'une rareté supérieure ou égale à rare.

6.3 Screens

Dans cette partie nous allons donner quelques exemples non exhaustifs d'impressions écrans de notre site internet.

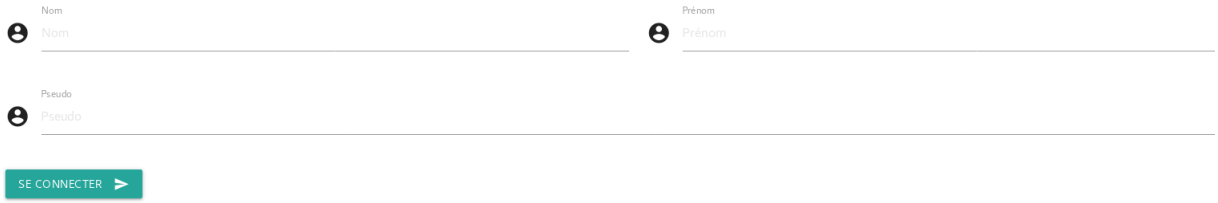

FIGURE 7 – Connexion

Commençons tout d'abord par la connexion sur la Figure 7, nous pouvons voir le bandeau de navigation permettant de naviguer vers les différents onglets de notre application (précédemment énoncés). Quand le joueur se connectera (le joueur test est 'Seto' 'Kaiba'), il sera redirigé vers la page de ses cartes.

Sur la Figure 10 nous pouvons voir la liste des cartes (exemplaires) du joueur qui est connecté. Grâce aux icônes à côté de chacune cartes, il peut au choix : consulter la carte, la supprimer de sa collection.

FIGURE 8 – Ajout d'un joueur

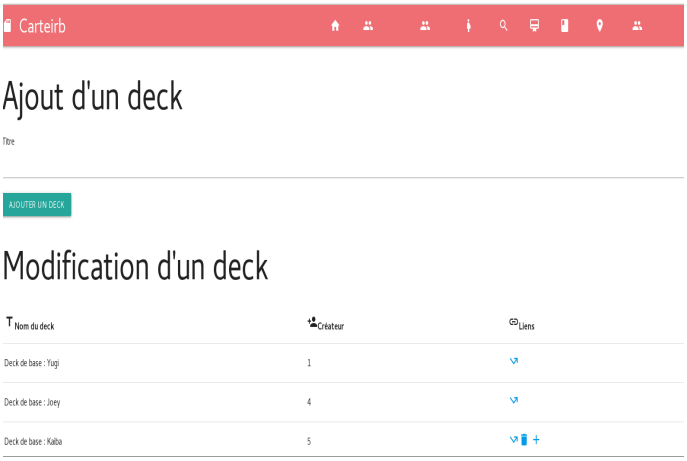

FIGURE 9 – Informations sur les decks

Sur la Figure 8 nous pouvons voir comment nous ajoutons un joueur (il faut remplir au minimum les champs noms et prénoms) ou bien nous pouvons accéder aux pages concernant les requêtes qui sont demandées.

Si nous accédons à la page concernant les cartes montrée sur la Figure 11, alors nous pouvons remplir un formulaire permettant d'ajouter une carte dans la base de données.

Si l'on a souhaité voir une carte en particulier, nous atterrissons sur la page sur la Figure 12. Sur cette dernière nous pouvons voir toutes les caractéristiques associées à une carte, supprimer ou ajouter une caractéristique (grâce à un formulaire).

Plus bas dans cette même page (comme sur la Figure 13, nous pouvons voir les infos concernant la carte à savoir le titre, le type, la famille et la nature de la carte.

Enfin, si nous pouvons avoir accès aux différents decks de la BDD comme montré sur la Figure 9. A travers cette page nous pouvons voir la composition de chaque deck grâce à l'icône correspondant et si nous sommes connecté en tant que joueur, nous pouvons modifier nos decks.

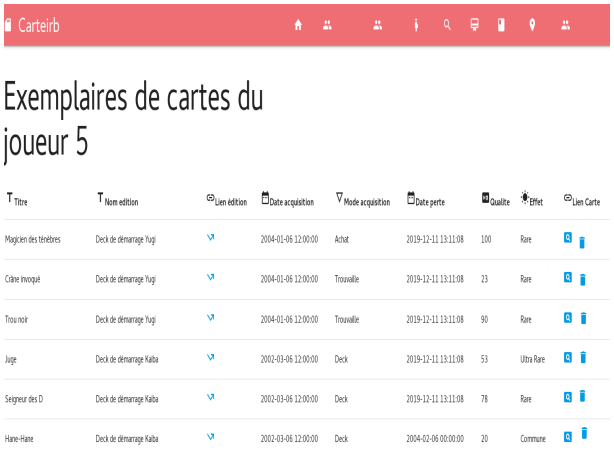

FIGURE 10 – Exemplaires du joueur 5

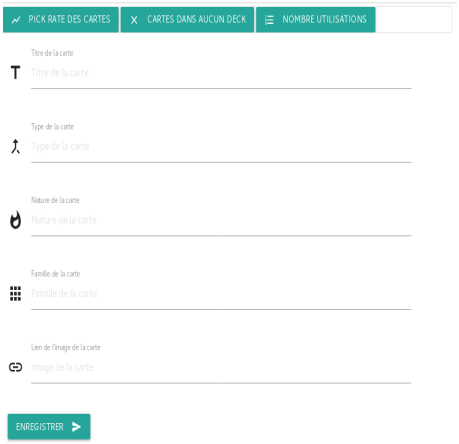

FIGURE 11 – Liste des requêtes demandées de cartes et formulaire

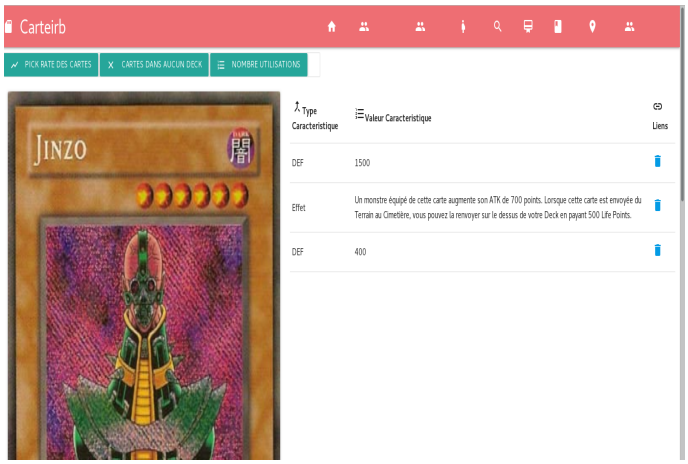

FIGURE 12 – Exemple d'affichage d'une carte

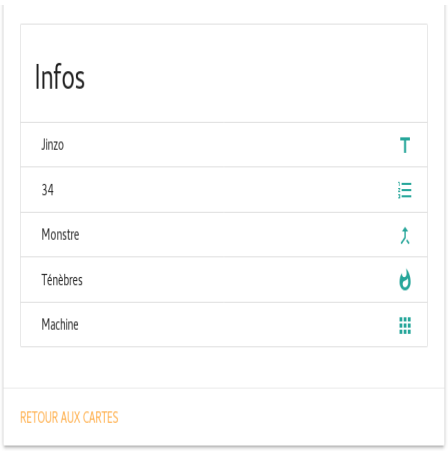

FIGURE 13 – Informations précises d'une carte

6.4 Features/Requêtes supplémentaires

Nous avons rajouté quelques *features* dans notre site et base de données.

- Liste des joueurs et l'adversaire le plus battu : `PlayerCounters.php`
- Liste des joueurs classés par leur nombre de défaites : `PlayerLosses.php`
- Liste des joueurs et le nombre de victoire contre chaque autre joueur : `PlayerNbVictoriesOpponent.php`
- Liste des joueurs classés par leur taux de victoires : `PlayerWR.php`
- Liste des cartes classées par leur taux de choix (pickrate) dans les decks : `Pickrate.php`

7 Conclusion

Ce projet comprend donc l'implémentation d'une base de données de cartes et son utilisation dans un site consultable. Nous avons découvert différents outils pour mettre en place une base de données. Nous avons du tester par exemple plusieurs langages (nous étions dans un premier temps parti sur PostGreSQL), pour ensuite faire notre choix. Le site à aussi vu plusieurs phase de développement dont une première sans *framework* (mais peu estéthique) puis une seconde, cette fois-ci avec un framework.