RAPPORT DE MI-PROJET

2022

PRÉPARÉ PAR

LUIS DOUDEAU
THOMAS FAUCHER
LENNY DE NARDI

SAE 3.02 - WEB : R3.01 BD R3.07

Grand Galop

IUT ORLEANS

Compte rendu

Introduction

Dans le cadre de notre cursus du BUT Informatique, nous sommes en charge de développer une application Web pour un client possédant un club de poney dans un village de Sologne. Nous disposons d'une partie mineure du système d'information et nous avons dû l'imaginer afin qu'il réponde de la meilleure manière au besoin du client. En effet, différentes contraintes ont été listées par le client sur ce système d'information, mais nous en avons implémenter de nombreuses autres que nous pensions essentielles. Dans un premier temps, nous avons développé une application dite native, fonctionnant en console. Puis, dans un second temps nous avons commencé à développer l'application Web finale que nous livrerons au client une fois terminée.

Choix des technologies utilisées

Tout d'abord, les technologies souhaitées par le client sont :

- Application Native en console avec Java et JDBC

Nous avons donc respecté les demandes du client.

De plus, Java et JDBC possèdent de nombreux avantages non négligeables : Java est un langage de programmation populaire et est utilisé à grande échelle dans le monde entier pour le développement d'applications. Il présente des avantages tels que le multithreading, l'extensibilité, la gestion de la mémoire, la haute sécurité, le support communautaire, etc.

En ce qui concerne JDBC, avec toutes les classes qu'il comporte, nous avons pu créer une application capable de se connecter au système de gestion de données. L'API JDBC est conçue pour être indépendante de tout SGBD. Donc, nos programmes pourront se connecter à n'importe quelle base de données en utilisant la même syntaxe.

Notre SGBD est MySQL, un système de gestion de bases de données que le client souhaite. MySQL est simple d'utilisation, performant, léger et gratuit.

I - Contraintes

1 - Dépendances fonctionelles

id → nomp, prenomp, ddn, poids, adressemail, adresse, code_postal, ville, numerotel,mdp
idpo → nomp, poidssup
idc → nomc, descc, typec, prix, id
(id, idpo, date_horaire) → idc, duree, a_paye
id (idClient) → cotisationA

Nous avons implémentés toutes ces contraintes grâce au mot-clé PRIMARY KEY de SQL, qui assure qu'un id (clé primaire) ou qu'un couple d'id (couple clé primaire) soit unique dans la table, autrement dit présent qu'une seule fois.

2 - Contraintes d'inclusion

MONITEUR(id) ⊂ PERSONNE(id)

La table MONITEUR est une table fille à la table PERSONNE, elle hérite de cette table. Donc l'id du MONITEUR doit être inclus dans la table PERSONNE pour exister.
Nous avons implémenté cette contrainte grâce au mot-clé FOREIGN KEY, qui assure que l'id de la table MONITEUR existe dans la table PERSONNE.

CLIENT(id) ⊂ PERSONNE(id)

Cette contrainte est similaire à celle du dessus.

COURS(id) ⊂ MONITEUR(id)

Cette contrainte d'inclusion signifie que pour un cours de la table COURS, on possède également un moniteur de la table MONITEUR. Nous avons implémentés cette contrainte grâce au mot-clé FOREIGN KEY, qui assure que l'id de la table COURS existe dans la table MONITEUR.

RESERVER(idpo) ⊂ PONEYS(idpo)
RESERVER(idc) ⊂ COURS(idc)
RESERVER(id) ⊂ CLIENT(id)

Ces trois contraintes d'inclusions ci-dessus sont issues de la dépendance fonctionnelle :

(id, idpo, date_horaire) → idc, duree, a_paye

C'est une association entre 3 entités : CLIENT, PONEYS et DATE_HORAIRE
Nous avons implémentés ces trois contraintes grâce au mot clé PRIMARY KEY et FOREIGN KEY de SQL, qui assurent qu'un couple de valeur *(id, idpo, date_horaire)* soit unique dans la table RESERVER et que chaque clé de ce couple clé primaire soit inclues dans leur table respective.

3 - Contraintes de vérification

Dans le sujet, diverse contraintes de vérification ont été citées :

- **Les clients doivent régler une cotisation annuelle.**
- **Les clients payent une cotisation aux cours qu'ils ont réservés.**

Ces deux contraintes de vérification signifient qu'une réservation ne peut pas être réalisée avec succès si le client qui l'effectue n'a pas payé sa cotisation annuelle ou s'il n'a pas payé le cours associé à cette réservation.

Pour implémenter cette contrainte, nous avons utilisé un trigger (verifPayement) qui se déclenche lors de l'insertion d'une nouvelle réservation dans la table RESERVER, et qui vérifie que le client a bien payé sa cotisation annuelle et son cours. Si c'est le cas, on l'insert. Sinon, un message d'erreur s'affiche et la réservation n'est pas insérer.

- **Un cours dure au maximum deux heures**

Nous avons implémenté cette contrainte avec un trigger (verifHeuresMaxCours) qui vérifie que la durée du cours ne dépasse pas 2 heures.

- **Les poneys doivent avoir au moins 1 heure de repos après 2 heures de cours.**

Cette contrainte est sûrement l'une des plus complexes à implémenter. Nous avons utilisé encore une fois un trigger (verifHeureRepos) pour l'implémenter dans notre base .
Dans celui-ci, nous vérifions si dans notre base il existe une réservation le même jour que celle que l'on souhaite insérer, avec le même poney. On parcourt ces réservations (si elles existent), puis nous allons vérifier pour chacune d'elles que si leur durée est de deux heures, la réservation que l'on souhaite insérer respecte le temps de repos d'une heure :

Admettons que cette réservation existe dans notre base :

Réservation le 24/11/2022 à 14h00 avec le poney 1 d'une durée de 2h

Voici le résultat si on souhaite insérer les réservations suivantes :

- Réservation le 24/11/2022 à 14h30 avec le poney 1 d'une durée de 2h : ERREUR

Cette réservation renvoie une erreur car, le poney est déjà en cours.

- Réservation le 24/11/2022 à 12h00 avec le poney 1 d'une durée de 2h : ERREUR

Cette réservation renvoie une erreur car le poney 1 enchaîne deux cours de deux heures sans repos. Or, il doit avoir au moins 1 heure de repos.

- Réservation le 24/11/2022 à 16h00 avec le poney 1 d'une durée de 1h : ERREUR

Cette réservation renvoie une erreur car le poney 1 enchaîne un cours de deux heures par un autre cours, sans repos. Or, il doit avoir au moins 1 heure de repos.

- Réservation le 24/11/2022 à 17h00 avec le poney 1 d'une durée de 2h : CORRECT

Cette réservation est insérée, car le poney a bien une heure de repos entre ses deux cours.

- **Seulement des cours particuliers ou collectifs avec 10 personnes maximum**

Pour cette contrainte de vérification, nous avons utilisé un trigger (ajoutPersonneCollectif)

Ce trigger compte le nombre de personnes au cours associé à la nouvelle réservation. Si c'est un cours particulier, et que le cours possède déjà une réservation, ou si c'est un cours collectif et que le nombre de réservations est déjà de 10, alors le trigger renvoie une erreur et la réservation n'est pas insérer dans la base.

- **Les poneys peuvent porter un cavalier jusqu'à un certain poids.**

Pour cette contrainte, nous avons besoin tout d'abord de posséder deux valeurs :

Le poids du client qui souhaite réserver un poney et le poids que peut supporter ce poney.

Nous stockons donc le poids des personnes dans la table PERSONNE, et le poids supportable par le poney dans la table PONEYS.

Lors d'une réservation, on vérifie avec des jointures que le poids du client est inférieur ou égal ou poids supportable par le poney. Si ce n'est pas le cas le trigger renvoie une erreur et la réservation n'est pas insérer.

Jusqu'ici, nous avons listé toutes les contraintes données par le client. Mais nous avons implémenté d'autres contraintes essentielles au bon fonctionnement de l'application, les voici :

- **Vérifier l'heure de réservation d'un cours entre 8h00 et 20h00 (verifHeureReservation)**

En effet, ils ne nous semblent pas logique de pouvoir réserver un cours avant 8h00 du matin et après 20h00 du soir. Nous avons donc implémenter un trigger permettant de vérifier l'heure de la réservation du cours afin de rendre réaliste nos données.

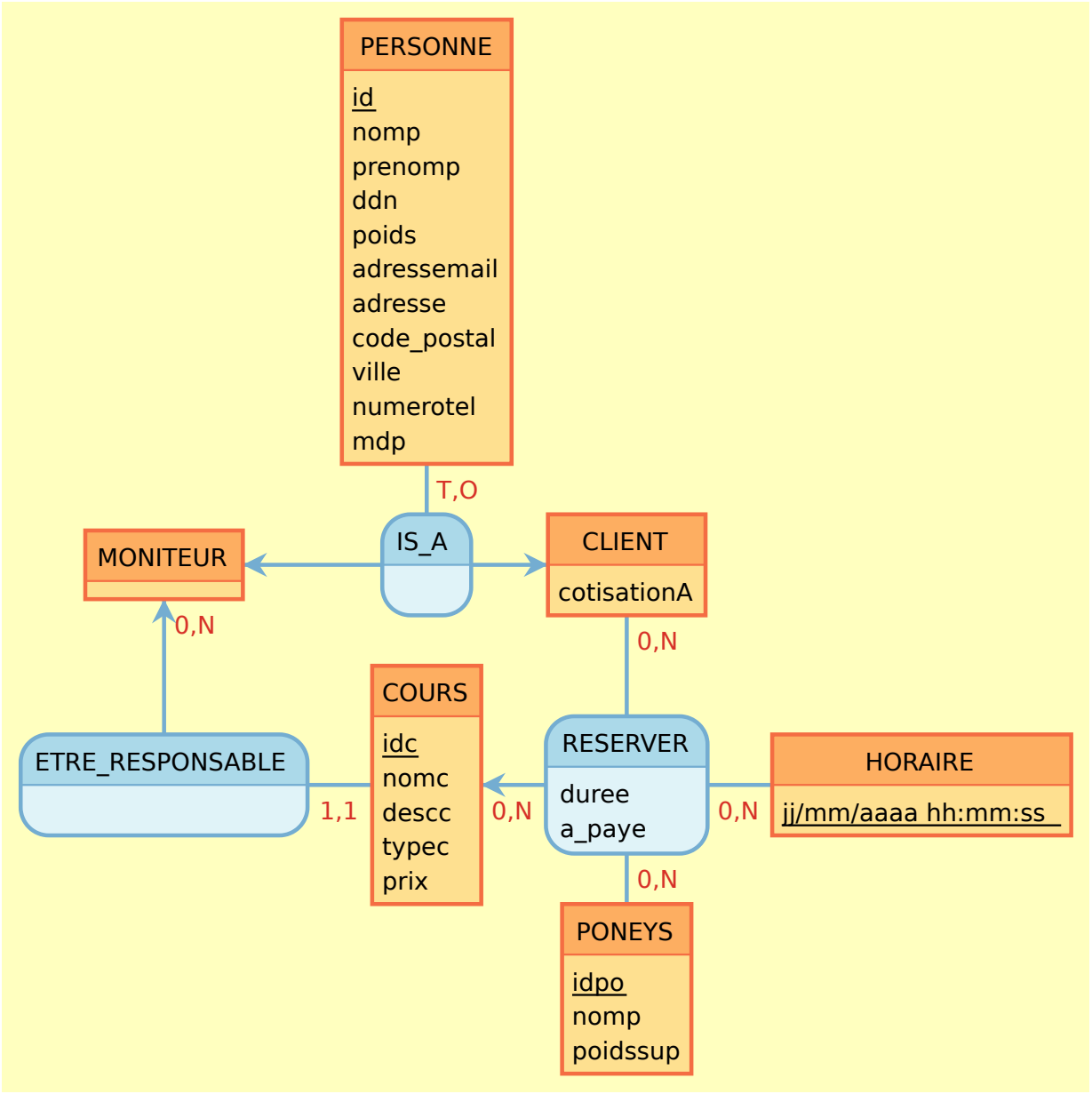
- **Vérifier qu'un client ne peut réserver qu'un cours à une même horaire**

Cette contrainte est essentielle au bon fonctionnement de l'application, elle permet de ne pas avoir plusieurs réservations qui se chevauchent pour un même client (un client ne peut pas suivre deux cours à la fois). Le trigger ajoutPersonneHoraire permet de vérifier lors de l'ajout d'une nouvelle réservation que le client associé à celle-ci ne soit pas déjà présent dans un autre cours au même horaires (ou horaires qui se chevauchent).

- **Vérifier qu'un moniteur ne peut être responsable que d'un seul cours à un même horaire**

Nous avons implémenté cette contrainte avec un trigger. Il permet de vérifier qu'un moniteur encadre un unique cours à la fois (ajoutMoniteurHoraire).

II - Modèle conceptuel des données (MCD)



III - Modèle logique des données (MLD)

PERSONNE [**id**, nomp, prenomp, ddn, poids, adressemail, adresse, code_postal, ville, numerotel, mdp]
MONITEUR [***id***]
CLIENT [***id***, cotisationA]
COURS [**idc**, nomc, descc, typec, prix, ***id***]
PONEYS [**idpo**, nomp, poidssup]
RESERVER [**jmahms, *id*, *idpo***, idc, duree, a_paye]

IV - Précision sur notre travail

Vous pourrez retrouver notre travail dans le dossier : RENDU_1_GRAND_GALOP_DOUDEAU.zip

Dans ce répertoire, se trouvent 3 autres répertoires :

- DEVELOPMENT
  - bin
  - img
  - scripts
  - src

Dans ce dossier, se trouve notre application native en console. Il y a dans le dossier 'scripts' le fichier de création de la base de données et le fichier d'insertion afin de peupler la base.
Dans le dossier '*src*' se trouve les fichiers Java (l'application elle-même).

Vous pourrez retrouver au début du fichier ***ExePonney.java***, des explications afin lancer notre application de la bonne manière et éviter de rencontrer des erreurs.

Dans le fichiers insPon.sql se trouve tout en bas dans l'insertion de la table RESERVER, des lignes d'insertions mise en commentaire. Elles vont vous permettre de voir que toutes les contraintes citées auparavant sont bien implémentées dans notre base. Il vous faut pour cela supprimer les commentaires une par une (attention, ce détail est important.). Puis relancer le script de création des tables insPon.sql et le fichier d'insertion insPon.sql et constater que la ligne décommentée n'est pas insérée, car elle ne respecte pas une des contraintes fixée sur la base. Vous devez commenter de nouveau la ligne, avant de supprimer le commentaire d'une autre.

- GESTION_PROJET

Dans ce dossier, vous pourrez retrouver notre MCD, un PDF avec les tâches effectuées par les différents membres du groupe ainsi que nos différentes maquettes de notre application WEB.

- flaskr

Ce dossier n'est pas utile pour le moment. Il contient toute l'arborescence de l'application WEB Flask, qui n'est pas encore entièrement terminée. Vous pouvez tout de même lancer l'application en exécutant le fichier run.py qui se trouve au chemin suivant : /flaskr/run.py

Une invitation à notre dépôt GITHUB vous à été envoyée, si vous souhaitez voir l'entièreté des commits.

Pour plus d'information, vous pouvez lire le README.md qui se trouve à la racine du projet. Il contient de nombreuses autres informations essentielles par rapport au projet. Sinon vous pouvez nous contacter via nos adresses mails universitaires.