Projet Base de Données
Compte rendu

Club de montagne

Adam BENOIT - Anaïs DOUET - Mathieu MEUNIER - Pierre POINAS - Steve LENING
Grenoble INP - Ensimag, UGA
Année 2023-2024

Contexte du projet

Un club d'activités de montagne souhaite mettre en place une base de données afin de permettre la gestion des services proposés à leurs adhérents et au grand public. Le service proposé est simple :

- l'ensemble des utilisateurs inscrits sur ce système ont la possibilité de faire des réservations de repas/nuits dans les refuges gérés par le club ;
- seuls les adhérents du club ont la possibilité de s'inscrire à des formations et de réserver du matériel pour leurs sorties en montagne.

Les caractéristiques du projet à implémenter sont décrites de manière plus exhaustives dans le sujet.

Table des matières

I. Analyse

mail_refuge (str)	$\rightarrow$	nom_refuge	(str)
	$\nrightarrow$	telephone_refuge	(str)
	$\rightarrow$	secteur_geographique	(str)
	$\rightarrow$	presentation	(str)
	$\rightarrow$	date_ouverture	(date)
	$\rightarrow$	date_fermeture	(date)
	$\rightarrow$	places_repas	(int)
	$\rightarrow$	places_nuit	(int)
	$\rightarrow$	prix_nuitee	(int)
mail_refuge	$\twoheadrightarrow$	type_moyen_paiement	(str)
mail_refuge	$\twoheadrightarrow$	type_repas	(str)
mail_refuge, type_repas (str)	$\rightarrow$	prix	(int)
id_formation (str)	$\rightarrow$	nom_formation	(str)
	$\rightarrow$	date_formation	(date)
	$\rightarrow$	duree_formation	(int)
	$\rightarrow$	description_formation	(str)
	$\rightarrow$	places_formation	(int)
	$\rightarrow$	prix_formation	(int)
id_formation	$\twoheadrightarrow$	type_activites	(str)
marque (str), modèle (str), date_achat (date)	$\rightarrow$	categorie	(str)
	$\rightarrow$	nb_pieces	(int)
	$\rightarrow$	prix_casse	(int)
	$\rightarrow$	description_materiel	(str)
	$\rightarrow$	date_peremption	(date)
marque, modèle, date_achat	$\twoheadrightarrow$	type_activites	(str)
categorie (str)	$\rightarrow$	categorie_parente	(str)
id_utilisateur (int)	$\rightarrow$	somme_due	(int)
	$\rightarrow$	somme_remboursee	(int)
	$\nrightarrow$	id_adherent	(int)
mail_membre (str)	$\rightarrow$	mot_de_passe	(str)
	$\rightarrow$	nom_membre	(str)
	$\rightarrow$	prenom_membre	(str)
	$\rightarrow$	adresse_membre	(str)
id_reservation_refuge (int)	$\rightarrow$	date_reservation_ref	(date)
	$\rightarrow$	heure_reservation_ref	(int)
	$\rightarrow$	prix_total_reservation	(int)
id_reservation_refuge, jour (int)	$\not\twoheadrightarrow$	type_repas	(str)
id_reservation_formation (int)	$\rightarrow$	rang_attente	(int)
id_reservation_materiel (int)	$\rightarrow$	nb_pieces	(int)
	$\rightarrow$	date_recup	(date)
	$\rightarrow$	date_rendu	(date)
	$\nrightarrow$	nb_casse	(int)

L'analyse des dépendances fonctionnelles s'est en grande partie assez naturellement. Cependant, quelques points on été plus complexes que d'autres, notamment l'arbre des catégories. Le choix fait pour représenter cette structure d'arbre, qui n'est pas implémentée en SQL2, a été de relier chaque catégorie à sa catégorie parente dans l'arbre. La racine sera choisie en ayant elle-même comme parent.

Dépendances fonctionnelles	Contrainte de valeur	Contrainte de multiplicité	Contraintes contextuelles
mail_refuge → nom_refuge → secteur_geographique → presentation → date_ouverture → date_fermeture → places_repas → places_nuit → prix_nuitee mail_refuge, type_repas → prix id_formation → nom_formation → date_formation → duree_formation → description_formation → places_formation → prix_formation marque, modèle, date_achat → categorie → nb_pieces → prix_casse → description_materiel → date_peremption categorie → categorie_parente id_utilisateur → somme_due → somme_remboursee mail_membre → mot_de_passe → nom_membre → prenom_membre → adresse_membre id_reservation_refuge → date_reservation_ref → heure_reservation_ref → prix_total_reservation id_reservation_formation → rang_attente id_reservation_materiel → nb_pieces → date_recup → date_rendu	$date\_ouverture \geq date\_fermeture$ $places\_nuitee \geq 0$ $places\_repas \geq 0$ $prix\_nuitee \geq 0$ $prix \geq 0$ $duree\_formation \geq 0$ $places\_formation \geq 0$ $prix\_formation \geq 0$ $nb\_pieces \geq 0$ $prix\_casse \geq 0$ $somme\_due \geq 0$ $somme\_remboursee \geq 0$ $prix\_total\_reservation \geq 0$ $rang\_attente \geq 0$ $date\_rendu \geq date\_recup$ $nb\_casse \geq 0$	Le refuge peut ne pas avoir de téléphone : $mail\_refuge \nrightarrow telephone\_refuge$ Un refuge a, au minimum, un moyen de paiement : $mail\_refuge \twoheadrightarrow type\_moyen\_paiement$ Un refuge propose au moins un type de repas : $mail\_refuge \twoheadrightarrow type\_repas$ Une formation comporte au moins une activité : $id\_formation \twoheadrightarrow type\_activites$ Un même matériel peut servir pour plusieurs activités : $marque, modèle, date\_achat \twoheadrightarrow type\_activites$ Un utilisateur peut au maximum être reconnu comme adhérent : $id\_utilisateur \nrightarrow id\_adherent$ Pour chaque jour de la réservation, on peut prendre plusieurs ou aucun repas : $id\_reservation\_refuge, jour \not\twoheadrightarrow type\_repas$ Il peut ne pas avoir eu de casse de matériel durant son utilisation : $id\_reservation\_materiel \nrightarrow nb\_casse$	• Il ne peut pas y avoir plus de réservation en même temps que de places dans un refuge. • La date de réservation d'un refuge est future (minimum aujourd'hui) et durant la date d'ouverture d'un refuge

FIGURE 1 – Tableau des Dépendances Fonctionnelles et de contraintes associées

II. Conception

1. Diagramme

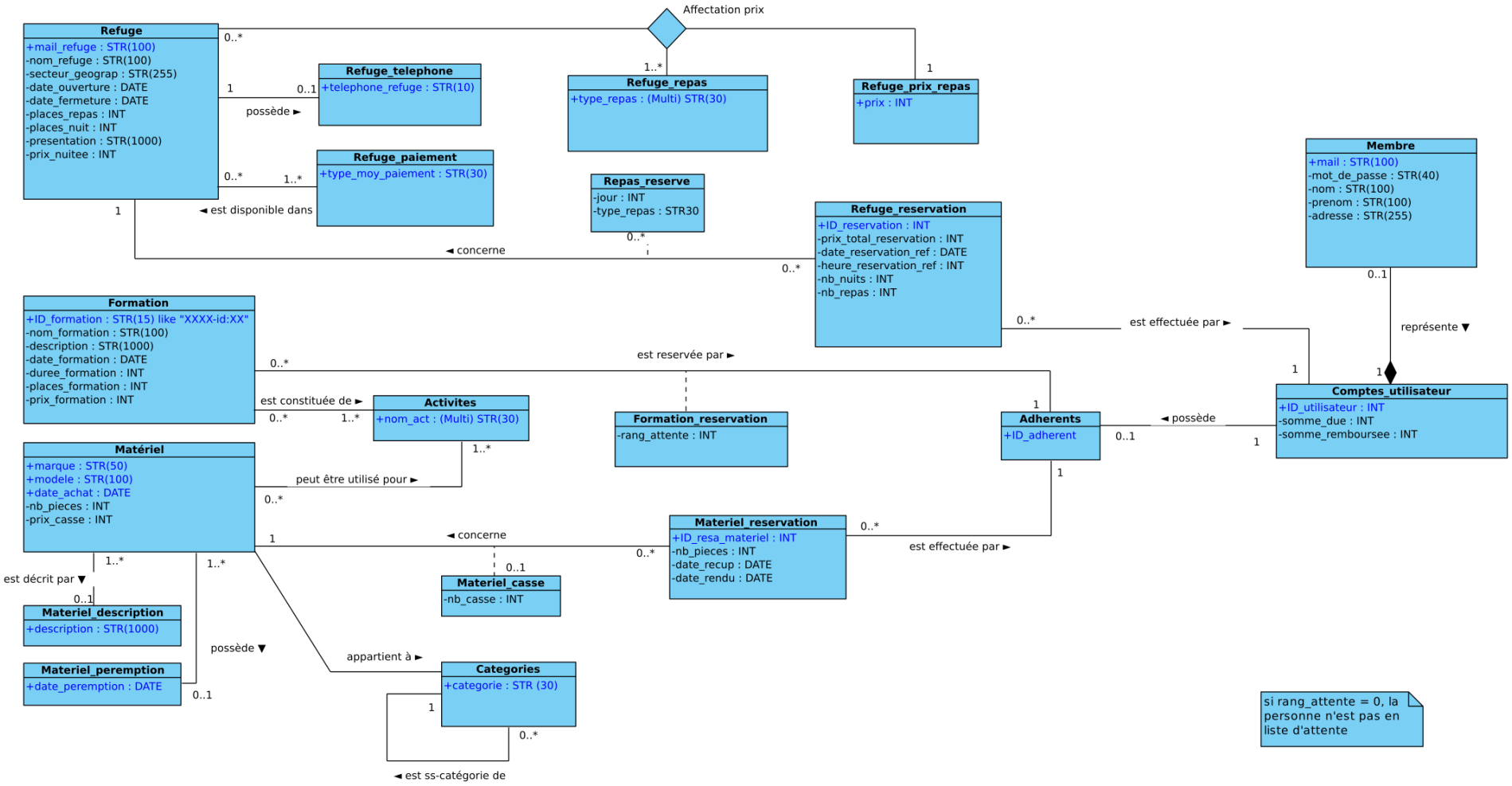

FIGURE 2 – Diagramme conceptuel de notre base de données

2. Difficultés rencontrées

Pour traduire les dépendances fonctionnelles en schéma relationnel, nous sommes passés par le diagramme entité-association (conceptuel) ci-dessus. C' est de loin la tâche qui nous a prit le plus de temps et le plus de réflexion. En effet, le premier diagramme réalisé était trop réflechi comme un schéma logique dès le début et il a été long de comprendre les erreurs de cette approche pour arriver au diagramme actuel.
On note que les membres sont des entités faibles des comptes utilisateurs. En effet, ceux-ci ne peuvent pas exister sans être liés à un compte. Cependant, les attributs des membres ne pouvaient pas simplement figurer dans l'entité Comptes_utilisateur pour des raisons de RGPD. Pour respecter le droit à l'oubli, il faut que ces attributs puissent être effacés simplement car ils représentent les informations personnelles du compte utilisateur.

L'arbre des catégories est traduit par une entité qui se référence elle-même, pour former justement cette structure d'arbre.
Les réservation de refuge sont associées aux comptes utilisateur tandis que les réservations de matériel et de formation sont associées aux adhérents, cela permet d'assurer que la réservation est bien effectué par un utilisateur qui adhère au système (qui a payé son adhésion).

III. Traduction en relationnel

1. Difficultés rencontrées

Du diagramme, la traduction a été plutôt simple. La difficulté de cette partie est venue des erreurs faites dans le diagramme conceptuel à la base. De ce fait, le diagramme changeant sans cesse pour corriger ces erreurs, il fallait régulièrement corriger également le schéma relationel (schéma logique). C'est le schéma ci-dessous que l'on a implémenté en sql.

2. Schéma relationel

Refuge (***mail_refuge***, nom_refuge, secteur_geograp, date_ouverture, date_fermeture, places_repas, places_nuit, presentation, prix_nuitee)
Refuge_paiement (***mail_refuge*** (Refuge), ***type_moyen_paiement***)
Refuge_telephone (***mail_refuge*** (Refuge), telephone_refuge)
Comptes_utilisateur (***id_utilisateur***, somme_due, somme_remboursee)
Membres (***mail_membre***, mot_de_passe, nom_membre, prenom_membre, adresse_membre, ID_utilisateur (Comptes_utilisateur))
Adherents (***ID_adherent***)
JointureAdherents (***id_utilisateur*** (Comptes_utilisateur), id_adherent (Adherents))
Refuge_reservation (***mail_refuge*** (Refuge), ***id_reservation***, date_reservation, heure_reservation, nb_nuit, nb_repas, prix_total, ID_utilisateur(Comptes_utilisateur))
Refuge_repas (***mail_refuge*** (Refuge), ***type_repas***, prix_repas)
Repas_reserve (***mail_refuge*** (Refuge_reservation), ***id_reservation*** (Refuge_reservation), ***jour***, ***type_repas***)
Formation (***ID_Formation***, nom_formation, description_formation, date_formation, duree_formation, nb_places, prix)
Formation_reservation (***ID_adherent*** (Adherents), ***ID_formation*** (Formation), rang_attente)
Categories (***categorie***, categorie_parente (Categories))
Materiel (***marque***, ***modele***, ***date_achat***, nb_pieces, prix_casse, categorie (Categories))
Materiel_description(description, ***marque*** (Materiel), ***modele*** (Materiel), ***date_achat*** (Materiel))
Materiel_peremption(date_peremption, ***marque*** (Materiel), ***modele*** (Materiel), ***date_achat*** (Materiel))
Activites (***type_activite***)
Formation_activites (***type_activite*** (Activites), ***ID_formation*** (Formation))
Materiel_activites (type_activites (Activites), ***marque*** (Materiel), ***modele*** (Materiel), ***date_achat*** (Materiel))
Materiel_reservation (***ID_reservation_materiel***, nb_pieces, date_recup, date_rendu, marque (Materiel), modele (Materiel), date_achat (Materiel), ID_adherent (Adherents))
Materiel_casse (***ID_reservation_materiel*** (Materiel_reservation), nb_casse)

IV. Requêtes & Transactions

Pour commencer, la plupart des requêtes sont simplement des requêtes permettant de récupérer les informations d'un refuge ou d'un utilisateur. Ces requêtes sont donc de simple 'SELECT' sur une table avec pour clé la clé

de l'élément qui nous intéresse.
En revanche un certain type de requêtes se démarquait, ce sont celles permettant la vérification de la quantité disponible d'un élément. Par exemple la requête :

```
SELECT count(id_reservation_ref) FROM Refuge_reservation WHERE mail_refuge=?
AND date_reservation_ref<? AND (date_reservation_ref+nb_nuits)>?
```

Cette requête va permettre de recupérer le nombre de places déjà prises pour une date précise. Elle récupère donc toutes les réservations du refuge voulu qui vont commencer avant la date recherchée et finir après.
Il y a aussi des requêtes permettant la création de certains éléments tels qu'une nouvelle réservation. Ces requêtes ont la particularité d'utiliser les séquences afin de créer facilement et de façon sécurisée ces nouveaux éléments. Par exemple :

```
INSERT INTO REFUGE_RESERVATION VALUES(?,seq_id_reservation_ref.NEXTVAL,?,?,?,?,?,?)
```

Comme on peut le remarquer cette requête permet de créer facilement un nouvel id de réservation de refuge tout en étant sûr qu'il n'existe pas déjà.
Finalement un dernier type de requête est le suivant :

```
UPDATE COMPTES_UTILISATEUR SET SOMME_DUE=SOMME_DUE+? WHERE ID_UTILISATEUR=?
```

Cette façon de mettre à jour des éléments permet d'incrémenter de façon sécurisée.

Pour ce qui est des transactions nous avons tout d'abord fais en sorte de vérifier plusieurs fois leur validité. Notamment juste avant de commencer à faire les transactions. De plus un savepoint est placé avant le début des transactions et si la transactions n'est pas valide elle est annulée et rollback.

V. Mode d'emploi du démonstrateur

Le démonstrateur se présente comme une classe java simple, fournie avec un Makefile (il suffit d'executer la commande 'make' depuis le répertoire bin) qui la compile et la lance.
L'interface de connexion s'affiche alors et demande une adresse mail et un mot de passe. Pour des fins de tests, il est possible de se connecter avec le couple 'admin'/'admin', mais certains utilisateurs simulés "réels" ont été créés également.
Une fois connecté, il suffit de naviguer dans les différents menus proposés en entrant les chiffres des options souhaitées. Il y a toujours une option pour revenir en arrière et on peut quitter l'application depuis le menu principal.
Les fichiers de création et de peuplage de la base de données sont présents mais ne sont pas lancés par le Makefile, ils doivent être lancés à la main.

VI. Bilan

1. Organisation

1. **Analyse statique :** [Tout le groupe]
   - Propriétés
   - Dépendances fonctionnelles
   - Détermination des contraintes (valeur, multiplicité, contextuelles)
2. **Analyse conceptuelle :** [Mathieu et Steve]
   - Entités importantes
   - Attributs d'associations
   - Détection des associations ternaires et entités faibles
3. **Traduction relationnelle :** [Anaïs, Pierre et Mathieu]
   - Migration des clés pertinentes
   - Traduction des associations ambiguës

4. **Implémentions des fonctionnalités avec JDBC :** [Adam, Steve]
   - Connexion à la base de données
   - Browsing de la base de données
   - Réservation (formation, refuge, matériel)
   - Droit à l'oubli
5. **Documents**
   - Diaporama [Anais / Tout le groupe]
   - Compte rendu [Mathieu et Pierre]

2. Point difficiles rencontrés

Le SGBD d'Oracle utilise le langage `PL/SQL`. Nous avons ainsi pu découvrir que la commande `IF EXISTS` n'est pas présente dans ce langage, contrairement à `MySQL`. Il n'est donc pas possible d'effectuer automatiquement tous les `DROP TABLE` (servant à réinitialiser la BdD) sans que Oracle génère une erreur si la table n'existe pas. Cependant, il s'agit seulement d'une sécurité servant à empêcher des erreurs non bloquante. Cela ne change en rien la base de données ni le projet en lui même, car on ne nous demande pas de réinitialiser automatiquement la base de données.
Le mot clé `SEQUENCE` de `PL/SQL` nous a bien servi à résoudre le problème de génération d'ID. En effet, pour générer une ID utilisateur, ou adhèrent, ou de réservation, nous voulions générer une séquence de chiffres qui s'incrémente d'elle-même. Cependant, le mot clé `AUTO_INCREMENT` n'existe pas sous Oracle. Nous avons donc créé une `SEQUENCE` qui s'incrémente (`INCREMENT BY 1`) à chaque `.NEXTVAL` ce qui nous a permis de résoudre le problème.