Projet : société de chemins de fer

NF18 - Printemps 2021

Groupe 1 :

Anaïs Débureaux (GI02)
Laurine Hamard (GI02)
Lucas Mabille (GI02)
Sacha Benarroch (GI02)

Note de clarification - Version 2

17/04/2021

Table des matières

1 Contexte et objectifs

Une société de chemins de fer souhaite développer une application permettant de gérer ses gares, ses trains, ses lignes de train ainsi que tout ce qui a trait aux billets qu'elle vend. Cette réalisation sera l'objet de notre projet dans le cadre de l'UV NF18.

2 Acteurs du projet

- Client : la société de chemins de fer
- Maître d'ouvrage : Youcef Amarouche
- La maîtrise d'oeuvre sera réalisée par l'équipe composée de :
  - Anaïs Débureaux
  - Laurine Hamard
  - Lucas Mabille
  - Sacha Benarroch

3 Énoncé du cahier des charges

La société souhaite gérer des gares, des trains et des lignes de train. Parmi les propriétés de ces éléments, elle souhaite notamment répertorier et gérer les horaires de passage de chaque train dans chaque gare, sur chaque ligne. À partir de là, elle souhaite également pouvoir gérer les billets qui sont achetés par ses clients, ainsi que lesdits clients. La société souhaite également fournir à ces clients la possibilité de consulter un itinéraire au choix ainsi que les horaires des trains qui permettent de le suivre. La gestion des billets doit également être implémentée. Enfin, la société souhaite avoir accès à des fonctionnalités plus poussées pour obtenir des informations globales de nature statistique sur son fonctionnement.

4 Contraintes technologiques

L'application que nous devons livrer doit être développée en Python. La base de données sera gérée par le SGBD *PostgreSQL*. L'intégralité des livrables doit être déposée sur le *Gitlab* de l'UTC.

5 Livrables à fournir et échéances

Le calendrier suivant a été fixé pour les différents livrables attendus :
— Premier jet du MCD et note de clarification : semaine du 05/04/2020.
— MCD corrigé en accord avec le maître d'ouvrage, MLD : semaine du 12/04/2020
— Implémentation fonctionnelle en SQL : semaine du 19/04/2020
— Rendu applicatif fonctionnel : semaine du 26/04/2020
— Rendu applicatif final : semaine du 03/05/2020

6 Appropriation du cahier des charges

6.1 Identification des classes

Après une analyse approfondie du cahier des charges, nous avons dégagé les classes qui devront être implémentées. Celles qui se sont avérées les plus évidentes à la lecture du sujet sont les suivantes :
— `Gare`
— `Train`
— `Ligne`
— `Billet`
— `Calendrier`
— `Voyageur`

Parmi ces classes, nous en avons identifié 3 comme étant abstraites :
— `Train` dont hérite un nombre indéfini de classes : autant qu'il sera nécessaire de gérer de types de train, assurément les classes `RER`, `TER` et `TGV`.
— `Calendrier` dont héritent les classes `RegulierC` et `ExceptionnelC` afin de gérer les calendriers réguliers et exceptionnels des trains.
— `Voyageur` dont héritent les classes `OccasionnelV` et `RegulierV`, afin de distinguer les voyageurs occasionnels des voyageurs réguliers.

Par suite, nous avons dégagé trois autres classes, vouées à jouer le rôle d'intermédiaires dans une association (il ne s'agit pas pour autant de classes d'association !) :
— `Trajet`, qui permettra de modéliser les caractéristiques d'un trajet effectué par un voyageur dans un train, d'une gare à une autre, avec un horaire de départ et un horaire d'arrivée.
— `Arret`, qui modélise un arrêt effectué sur une ligne à un certain horaire.
— `Horaire`, qui modélise les horaires de départ et d'arrivée d'un train lorsqu'il marque un arrêt.

6.2 Attributs et contraintes associées aux classes

Les classes décrites ci-dessus sont caractérisées par les éléments suivants :

— Une gare possède un nom, une ville, une adresse et une zone horaire. Elle est caractérisée par le couple clé `(nom,ville)`.
— Une ligne possède un nom et un code unique (clé de la relation).
— Un train possède un numéro unique (clé de la relation), un nombre maximal de places, une vitesse maximale et un booléen indiquant la présence ou non d'une première classe dans le train. Nous supposons que tous les trains comportent par défaut une seconde classe. Les classes héritées de la classe `Train` ne possèdent aucun attribut propre.

> **?** Le nombre maximal de places, la présence d'une première classe et la vitesse maximale d'un train sont déterminées par la sous-classe à laquelle appartient le train. Chacun de ces attributs prend donc une valeur constante, définie par la sous-classe et est propre à cette dernière. Plutôt que de représenter ces attributs avec leur valeur propre dans chacune des sous-classes (`RER`, `TGV`, `TER`), ce qui paraîtrait redondant, nous avons choisi de factoriser ces attributs dans la classe mère abstraite et d'exprimer ceci par la mention `const` sur les attributs concernés.

— Un arrêt est caractérisé par son rang sur la ligne.
— Un trajet est caractérisé par une place occupée dans un train, l'id du billet, un arrêt de départ et un arrêt d'arrivée.
— Un billet est décrit par un id (clé artificielle de la relation), une gare et une heure de départ, une gare et une heure d'arrivée, un prix et un booléen indiquant si une assurance a été prise ou non sur le billet. La classe comporte 1 méthode permettant de calculer la durée totale du voyage. Le billet est considéré modifiable (du point de vue conceptuel) si une assurance a été souscrite.
— Un horaire est constitué d'un horaire de départ et d'un horaire d'arrivée. Si le train ne s'arrête pas à l'arrêt alors arrivee=depart.
— Un calendrier régulier est modélisé par un tableau de 7 booléens indiquant pour chaque jour de la semaine s'il est applicable. Un calendrier exceptionnel est caractérisé par 2 dates : le premier et le dernier jour où il s'applique. Ces dates peuvent être égales si le calendrier ne concerne qu'un seul jour. Un calendrier exceptionnel possède aussi un booléen ajout qui vaut 1 si le calendrier concerne des trains ajoutés par rapport au calendrier normal et 0 sinon. Insistons sur le fait que la classe `RegulierC` concerne tous les calendriers récurrents (y compris les horaires particuliers d'un train le dimanche, mais applicables **tous** les dimanches e.g.).

> **?** La représentation des calendriers réguliers par un tableau de booléens est pratique. Il n'est cependant pas certain que cette modélisation soit viable en SQL, nous n'hésiterons pas à la revoir si nécessaire.

— Un voyageur est caractérisé par un id (clé artificielle de la relation), un nom, un prénom, une adresse et un numéro de téléphone. Un voyageur occasionnel n'est caractérisé par aucune information supplémentaire, alors qu'un voyageur régulier est caractérisé par son numéro de carte et son statut.

6.3 Identification des associations et classes d'association

Les associations qui ont été identifiées entre les classes décrites ci-dessus sont les suivantes :

- `circule sur` entre `Train` et `Ligne` : un train ne circule que sur une ligne, d'où une association `(1:N)`.
- `circule a` entre `Train` et `Horaire` : un train est associé à plusieurs horaires (un par arrêt qu'il dessert, et même plusieurs pour chaque arrêt qui peut être desservi plusieurs fois dans la même journée), et un horaire ne peut être respecté que par un seul train car la classe `Horaire` était une classe association entre `Train` et `Arret`, que nous avons décidé de transformer afin de pouvoir la lier à d'autres classes. D'où l'arité `(1:M)`.
- `se fait à`, entre `Horaire` et `Arret` : un arrêt peut être desservi à plusieurs horaires différents, mais un horaire ne peut correspondre qu'à un arrêt (encore une fois car Horaire était une ancienne classe association entre `Train` et `Arret`), d'où une association `(1:N)`.
- `se situe sur` entre `Arret` et `Ligne` : sur une ligne se situent plusieurs arrêts, alors qu'un arrêt ne peut se situer que sur une ligne.

  ? Nous concevons une instance de la classe `Arret` comme étant dédiée à une ligne. Un arrêt peut être desservi par n lignes mais dans ce cas, n tuples seront instanciés et liés à la même gare dans la table `Arret` : un pour chaque ligne.

- `se fait dans`, entre `Arret` et `Gare` : un arrêt se fait dans une seule gare, et dans une gare peuvent se faire plusieurs arrêts, d'où une association `(1:N)`.
- `depart` et `arrivee` entre `Horaire` et `Trajet` : tout trajet n'a qu'un seul horaire de départ et un seul horaire d'arrivée, mais un horaire peut concerner plusieurs instances de `Trajet`, d'où l'arité `(1:N)` pour ces deux relations.
- `correspond` entre `Trajet` et `Billet`. Pour les mêmes raisons de spécificité de la classe `Trajet`, et puisqu'un billet peut comporter plusieurs trajets, l'association est d'arité `(1:N)`.

  ? Nous supposons qu'un billet peut être acheté pour plusieurs trajets disjoints. Nous ne vérifierons donc pas que la gare d'arrivée d'un trajet correspond à la gare de départ d'un autre trajet dans le même billet.

- `concerne` entre `Horaire` et `Calendrier`. Un horaire ne correspond qu'à un calendrier spécifique, mais un calendrier peut correspondre à plusieurs horaires, d'où l'arité `(1:N)`.
- `contient` entre `Billet` et `Reservation`, d'arité `(1:1)`. À un billet ne sera associé qu'un moyen de paiement (espèces, CB ou chèque) d'ou l'arité.
- `est faites par` entre Reservation et Voyageur. On a considéré qu'une reservation ne pouvait être effectué que par un seul voyageur. D'autre part un voyageur peut effectué plusieurs réservations.
- `propose` entre Transport, Hotel, Taxis, et Reservation d'arité (N :M). À chaque reservation on associe plusieurs services, que ce soit des hotels, des taxis ou des moyens de transport. Chaque service peut être proposé pour plusieurs réservations.

6.4 Rôles des utilisateurs et fonctions à implémenter

Deux grandes catégories d'utilisateurs de notre application ont été identifiées, ainsi que les fonctionnalités à implémenter pour celles-ci :

— Un gestionnaire peut consulter, créer et supprimer tout ce qui a trait à la circulation des trains : gares, arrêts, trains, lignes, calendriers. Mais il doit aussi avoir la main sur les billets et les informations concernant les voyageurs afin d'offrir à ces derniers un service de qualité supérieure. Il doit enfin avoir accès à des statistiques concernant la société.

— Un voyageur doit pouvoir consulter les trajets réalisables ainsi que leurs horaires selon des critères (de prix, e.g.), acheter un billet, le modifier, le supprimer et accéder aux billets qu'il a réservés.

6.5 Justification des choix pour la MLD

Les transformations que nous avons décidées de faire pour les classes issues d'héritages sont les suivantes :

— un héritage par la classe mère pour `Voyageur`. En effet, l'héritage est presque complet. La classe mère est abstraite, la classe RegulierV a quelques attributs propres, toutes les associations sont avec la classe mère. Un attribut supplémentaire de domaine booléen est ajouté à la classe mère, afin de distinguer les tuples.

— - un héritage par la classe mère pour `Train` et la traduction des contraintes sur les attributs constants. L'héritage est complet c'est à dire que les classes filles ne définissent pas d'attributs autres que ceux hérités, cette solution est donc optimum. La classe Train est abstraite, les classes filles n'ont pas d'associations propres. Un attribut supplémentaire de discrimination type, est ajouté à la classe mère, afin de distinguer les tuples.

— - un héritage par classes filles pour `Calendrier`. L'héritage n'est pas complet, on exclut a priori l'héritage par la classe mère. Les classes filles ont leurs propres attributs mais pas leur propres associations. La classe mère est abstraite donc on n'exclut pas un héritage par classe fille s'il n'y a pas d'association problématique. L'héritage par référence implique deux contraintes complexes et nécessite deux classes de plus. On va donc préférer l'héritage par classes filles. Nous sommes dans un cas problématique de cet héritage avec association 1 :N (entrante) sur la classe mère avec la classe `Horaire`. La solution consiste à ajouter autant de clés étrangères que de classes filles et à gérer le fait que ces clés ne peuvent pas être co-valuées.

Les autres modifications que nous avons effectué sont les suivantes :

— -Nous avons ajouté un attribut "id" à la classe Horaire pour ne pas avoir à mettre trop attributs dans la clé, ce qui compliquerait les référencements d'un objet `Horaire` dans une autre table.

— -Pour les mêmes raisons, nous avons introduit un identifiant dans les classes filles de la classe `Calendrier`.