Rapport de NF18

La gestion des trains

Table des matières

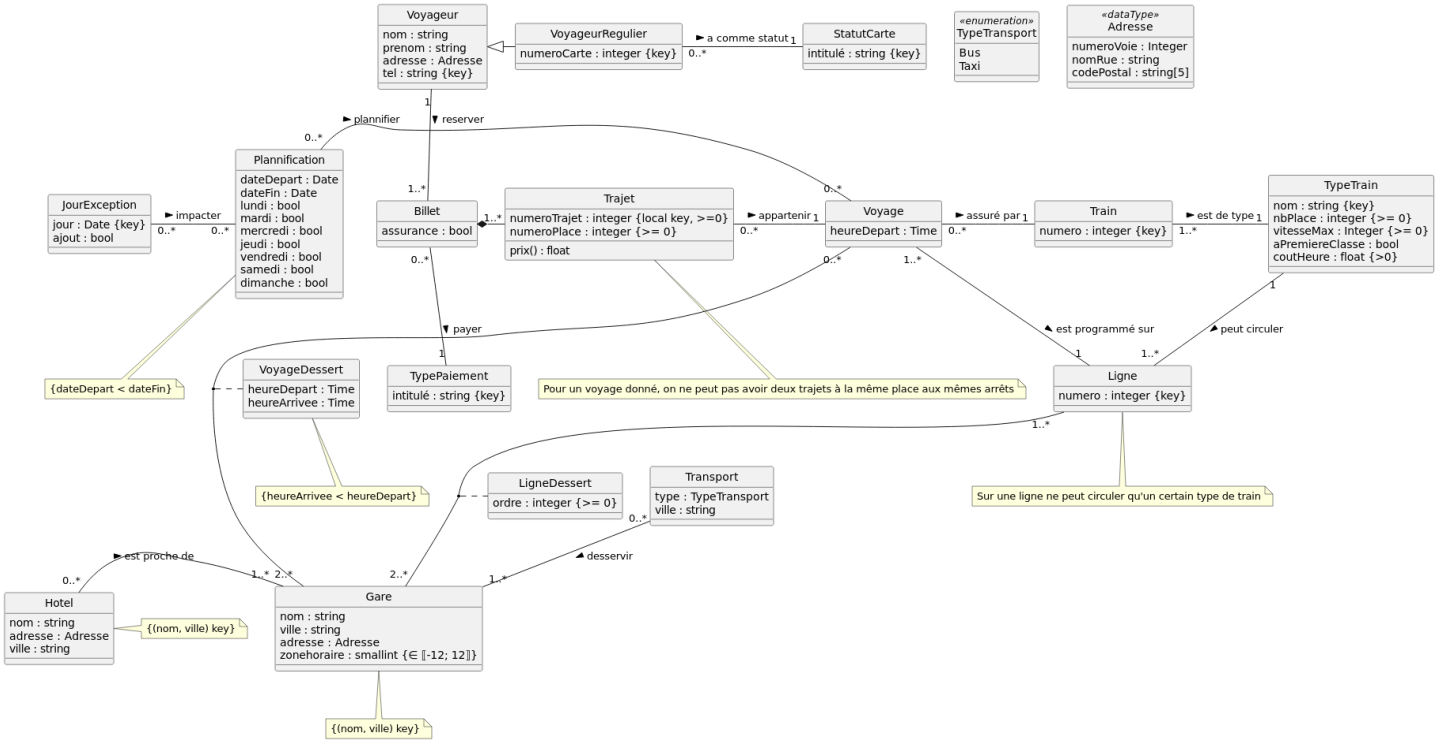
Introduction : les consignes du projet

Le projet sur lequel on a travaillé a consisté en la réalisation d'une application de gestion de train pour le compte d'une société de chemin de fer qui souhaitait mettre en place un système informatisé pour gérer ses gares, ses lignes, ses trains et ses billets. Notre mission a donc eu pour but de concevoir l'architecture de la base de données permettant de répondre au mieux à leurs besoins ainsi que de développer l'application permettant aux différent.e.s utilisateurs et utilisatrices d'interagir avec le système de gestion de données afin de convenir aux besoins de tout le monde.

I - La conception

Pour rappel, le G6 a été largement modifié après le premier rendu, ainsi le UML1 ne sera pas expliqué dans ce rapport.

I.A - UML



La composition Trajet - Billet :

On a décidé que l'association entre Trajet et Billet est une composition. Tout d'abord, cela a été guidé par l'information « Un billet peut être composé de plusieurs trajets » présente dans le sujet. Mais de manière plus précise, les caractéristiques d'une composition correspondaient exactement à cette association :

- La composition associe une classe composite et des classes parties, tel que tout objet partie appartient à un et un seul objet composite. C'est donc une association 1:N (voire 1:1).
    - Ici, tout trajet appartient bien à un, et un seul billet
- Le cycle de vie des objets parties est lié à celui de l'objet composite, donc un objet partie disparaît quand l'objet composite auquel il est associé disparaît.
    - Si le billet est supprimé de la Base de donnée, effectivement, les trajets qui le composent doivent aussi être supprimé

L'héritage Voyageur- Voyageur régulier :

D'après le sujet : « Il existe deux types de voyageurs : les voyageurs occasionnels et les voyageurs réguliers, qui ont une carte numérotée et un statut (bronze, silver, gold, platine...) ». En analysant, on réalise que les voyageurs réguliers ne sont qu'un type particulier de voyageurs occasionnels. De manière plus précise, un voyageur régulier a seulement des attributs supplémentaires par rapport aux voyageurs occasionnels (une carte numérotée et un statut (bronze, silver, gold, platine...)). Ainsi, il n'y a pas besoin d'avoir une classe mère abstraite, il suffit d'avoir comme classe mère voyageur qui désignera les voyageurs de manière général, et comme classe fille voyageur régulier.

De cette manière, les voyageurs particuliers hériteront de toutes les associations et attributs de la classe voyageurs (occasionnels) en plus d'avoir leurs attributs propres.

Classe association LigneDessert :

Étant donné qu'une gare est desservie par plusieurs lignes, et qu'une ligne dessert plusieurs gares, on se trouve dans une association N : M. De ce fait, il nous faut créer une classe association. mais aussi, les lignes desservent les gares dans un ordre précis et unique pour chaque gare :

- La ligne 1 dessert la Gare x en premier (ordre 1)
- La ligne 1 dessert la gare y en deuxième (ordre 2)
- ...
- La ligne 1 dessert la gare n en n-ième (ordre n)

Ainsi, on observe un attribut ordre dans la classe association Ligne. Le triplet (gare, ligne, ordre) devra être unique.

Classe association VoyageDessert :

Étant donné qu'une gare est desservie par plusieurs voyages, et qu'un voyage dessert plusieurs gares, on se trouve dans une association N : M. De ce fait, il nous faut créer une classe association. Mais aussi, les voyages desservent les gares dans à des heures précise (heure d'arrivée en gare et heure de départ de la gare) :

- Le voyage 1 arrive en gare x à 18h40 et part de la gare x à 18h43
- Le voyage 2 arrive en gare y à 20h20 et part de la gare x à 20h30
- ...

Ainsi, il est nécessaire que la classe association VoyageDessert ait pour attribut HeureArrivée et HeureDepart.

Classes StatutCarte et TypePaiment :

Pour éviter l'écueil de mise à jour trop fréquente, il n'est pas souhaitable de créer une énumération pour des moyens de paiements (et statut carte) qui évoluent rapidement dans le temps. Il n'est pas non plus souhaitable d'en faire un attribut typé string, car cela pourrait engendrer des erreurs. Alors, le plus adéquat semble être de créer une table.

Classe TypeTrain :

Étant donné que le type de train détermine de nombreux attributs, il a été nécessaire d'en faire une classe à part entière. Cette classe est ainsi associée à des trains dans une relation (N:1) et aux lignes qui ne laissent circuler qu'un seul type de train (N:1).
Remarque : on a ajouté un attribut coutHeure. En effet, on a supposé que la meilleure manière de calculer le coût d'un trajet, c'est de prendre le prix du type de train que l'on multiplie au nombre d'heures du trajet.

I.B - MLD

Transformation de l'héritage (Voyageur-VoyageurRégulier)

Comme pour tout héritage, on a trois possibilités :

1. Héritage par classe mère
2. Héritage par classe fille
3. Héritage par référencement

Et voici les informations que l'on a pour nous aider à choisir la manière dont on va transformer l'héritage :

- La classe mère n'est pas abstraite, il sera ainsi nécessaire d'instancier des voyageurs qui ne sont pas réguliers (dits occasionnels)
- L'héritage est presque complet, la classe fille ne possède que deux attributs supplémentaires
- La classe mère a des associations générales, la classe fille n'a pas d'association particulière

À la lumière de ces informations, la méthode la plus satisfaisante pour un passage au MLD semble être un héritage par la classe mère.

```
Voyageur(#idVoyageur : INTEGER, nom : STRING, prenom : STRING, numeroVoie : SMALLINT, nomRue : STRING, codePostal : STRING[5], tel : string, numeroCarte: INT, statut =>StatutCarte(intitulé))
   avec tel UNIQUE, numeroCarte NULLABLE UNIQUE, statut NULLABLE, ((numeroCarte IS NULL AND statut IS NULL) OR (numeroCarte IS NOT NULL AND statut IS NOT NULL))
```

Remarque : Pour avoir un accès plus simple aux voyageurs réguliers et aux voyageurs occasionnels, on a créé deux vues :

```
VvoyageurRégylier=Projection(Restriction(Voyageur, statut IS NOT NULL) idVoyageur, nom, prenom, numeroVoie, nomRue, codePostal, tel, numeroCarte, statut)

VvoyageurOccasionnel=Projection(Restriction(Voyageur, statut IS NOT NULL) idVoyageur, nom, prenom, numeroVoie, nomRue, codePostal, tel)
```

Gérer les associations

Les tables associations

De manière générale, il a fallu créer des tables associations pour chacune des associations N : M. Soit :

- Transportdessert pour la relation N : M entre gare et transport
- estProche pour la relation N : M entre gare et hotel
- LigneDessert par rapport à la classe association correspondante
- VoyageDessert par rapport à la classe association correspondante
- Impacte pour la relation N : M entre exception et planification
- Planifie pour la relation N : M entre planification et voyage

Les clefs étrangères pour les relations 1 : N

Pour toutes les relations 1 : N, on a ajouté à la relation côté N une clé étrangère vers la relation côté 1. Par exemple, dans la table Ligne, on retrouve une clef étrangère type référençant typeTrain.

Le choix des clefs

- Hôtel : On considère qu'il n'y a pas deux hôtels d'une même ville ayant le même nom, c'est donc NOT NULL et UNIQUE.

- Ligne : il nous a semblé cohérent que chaque ligne ait numéro unique et NOT NULL, ce n'est pas simplement une clef artificielle, car elle a aussi un rôle pour les usagers du site qui peuvent se référer au numéro de ligne.

- Transport : Vu qu'il n'y a pas de clef naturelle, on a dû créer une clef artificielle

- Voyage : Bien que heuredeDepart, train soient un couple UNIQUE, on ne souhaite pas qu'ils soient immuables, on pourra ainsi changer l'heure ou le train pour le même voyage. Il faut en conséquence créer une clef artificielle pour identifier un voyage

- Voyageur : un numéro de téléphone est certes unique et non nulle, il n'est cependant pas immuable pour un même voyageur, la clef artificielle semble se dessiner comme la meilleure option

II - (P)SQL

II-A La création de tables

Il n'a pas grand-chose à dire sur ce sujet, en effet le passage du MLD au SQL et plus mécanique qu'autre chose.

Comme seule remarque la création de la VPrixBillet et vNbPersParTrain, elles nous permettront de faire certaines requêtes plus facilement.

II-B Insertion de valeurs

Encore une fois, ce fut très mécanique comme étape, bien qu'il faille faire attention à la cohérence des données qui ne seront gérées que dans l'applicatif. Par exemple que l'ordre des gares desservies correspondent à l'heure des gares desservies par un voyage appartenant à cette ligne.

II-C Les requêtes

Pour les requêtes, elles sont nombreuses et de complexité différentes, en voici une brève explication :

Recherche des voyages disponibles de Paris à Compiègne le dimanche 05/03/2023 :

- La requête utilise plusieurs jointures pour récupérer les informations nécessaires.
- Elle vérifie la correspondance des gares de départ et d'arrivée, ainsi que la date spécifiée.
- Elle prend également en compte les exceptions de planification pour les jours spécifiques.

Les voyages disponibles sont sélectionnés en fonction des critères spécifiés.

Calcul du prix moyen d'un billet acheté :

- La requête utilise la fonction AVG pour calculer la moyenne des prix des billets dans la table VPrixBillet.

Calcul du prix moyen d'un billet selon le moyen de paiement :

- La requête effectue une jointure entre les tables VPrixBillet et Billet en utilisant l'ID du billet comme clé étrangère.
- Elle regroupe les résultats en fonction du type de paiement et calcule la moyenne des prix des billets pour chaque type.

Calcul du prix moyen d'un billet payé par statut :

- La requête effectue une jointure entre les tables VPrixBillet, Billet et Voyageur pour obtenir les informations sur le statut du voyageur.

Elle exclut les enregistrements avec un statut nul.
Les résultats sont regroupés par statut et le prix moyen des billets est calculé pour chaque statut.
Comptage du nombre de trajets effectués par un voyageur pour un voyage donné à une date et heure spécifiques :

La requête compte le nombre de lignes dans la table Trajet qui correspondent aux critères spécifiés.

- Elle utilise l'opérateur COUNT pour obtenir le nombre de trajets.

Récupération du nombre de places par voyage :

- La requête effectue plusieurs jointures entre les tables voyage, ligne et TypeTrain pour obtenir les informations sur le nombre de places disponibles.
- Les résultats incluent l'ID du voyage et le nombre de places du train correspondant.

Récupération du nombre de places pour un voyage spécifique :

- La requête est similaire à la précédente, mais elle ajoute une condition pour filtrer les résultats en fonction de l'ID du voyage spécifié.
-

Calcul du nombre de places restantes pour un voyage donné à une date et heure spécifiques :

- La requête effectue des jointures entre les tables trajet, voyage, ligne et TypeTrain pour obtenir les informations nécessaires.
- Les résultats incluent le nombre de places occupées, le nombre de places disponibles et le nombre de places restantes pour le voyage spécifié.

Récupération des voyages d'un voyageur :

- La requête récupère les informations sur les voyages d'un voyageur spécifique en utilisant des jointures entre les tables voyage, Trajet et Billet.
- On récupère les informations relatives à l'acheteur ayant pour id=1
- Les résultats incluent l'heure de départ, la ligne et le train pour chaque voyage.

Comptage du nombre de trajets effectués par un voyageur :

- La requête compte le nombre de lignes dans la table Trajet qui correspondent à l'ID du voyageur spécifié.

Comptage du nombre d'hôtels proches des gares :

- La requête compte le nombre de lignes dans la table HotelProcheDeGare.
- Les résultats sont groupés par le nom de la gare.

Calcul du taux de remplissage des trains :

- La requête effectue des jointures entre les tables vNbPersParTrain, train et TypeTrain pour obtenir les informations nécessaires.
- Elle calcule le taux de remplissage en divisant le nombre de personnes par le nombre de places disponibles.

Récupération des gares par niveau de fréquentation dans l'ordre décroissant :

- La requête utilise la fonction COUNT pour compter le nombre de fois où chaque gare apparaît dans la table VoyageDessert.
- Les résultats sont classés par ordre décroissant de fréquentation.

Récupération des lignes les plus empruntées dans l'ordre décroissant :

- La requête utilise la fonction COUNT pour compter le nombre de fois où chaque ligne apparaît dans la table voyage.
- Les résultats sont classés par ordre décroissant du nombre d'emprunts.

III - PSQL et application

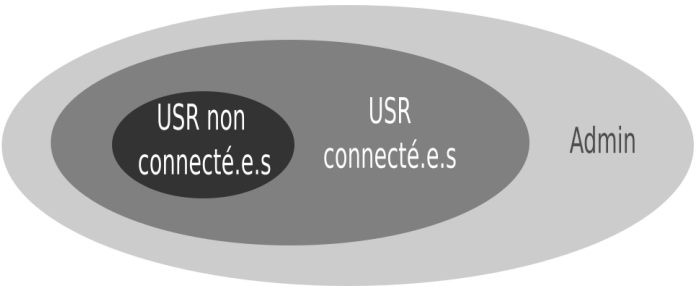
Afin de pouvoir interagir avec notre base de données, il nous a fallu concevoir et mettre au point une application permettant de répondre aux besoins des différents utilisateurs et utilisatrices selon ce qui nous était donné dans le cahier des charges.

III.A. Identification des rôles

Nous avons donc commencé par identifier les personnes destinées à utiliser notre application ainsi que leurs besoins. En consultant le cahier des charges, on a identifié trois utilisateurs différents :

- les admin qui peuvent :
    - ajouter, modifier, supprimer des lignes, des trains et leurs itinéraires
    - obtenir des statistiques sur le fonctionnement de la société
- les utilisateurs/utilisatrices connecté.e.s à l'application qui peuvent :
    - réserver des billets
    - annuler ou modifier leurs réservations
- les utilisateurs/utilisatrices non connecté.e.s qui peuvent :
    - consulter les horaires des trains
    - rechercher des trajets
    - se connecter pour réserver des billets

Ces rôles sont "emboités" les uns dans les autres dans le sens où les utilisateurs/utilisatrices connecté.e.s peuvent faire ce que les utilisateurs/utilisatrices non connecté.e.s peuvent faire. On peut ainsi représenter ces relations d'emboîtement ainsi :



Une fois les différents rôles de notre application bien définis, nous avons pu nous atteler à la tâche de développer cette dernière et d'écrire les requêtes SQL et les vues nécessaires à son bon fonctionnement.

III.B. Architecture de l'application

Pour chaque grand menu de l'application, il va être possible de naviguer de page en page en entrant dans la console un entier qui correspond à celui indiqué devant le nom du menu. Par exemple, sur le menu d'accueil (voir figure suivante), si l'on souhaite continuer sans se connecter, il faut rentrer le chiffre 0 dans la console.

III.B.1. Menu d'accueil

Au démarrage de l'application, c'est un écran d'accueil qui est affiché à l'écran.

```
/--------------------------------------\
|                                      |
|   Application de gestion de trains   |
|                                      |
\--------------------------------------/

Bienvenu !

------
0 - Utilisateur non connecté
1 - Connexion
2 - Admin
3 - Quitter l'application

> 
```

C'est à partir de cet écran que l'on va se connecter à l'application selon son rôle et de ce que l'on souhaite y faire.

III.B.2. Utilisateur/utilisatrice non connecté.e

Lorsque l'on indique que l'on souhaite continuer comme utilisateur/utilisatrice non connecté.e, on peut donc soit consulter les horaires des trains qui partent d'une certaine gare (2), soit consulter des trajets (1).

```
------
0 - Consulter les horaires des trains
1 - Chercher des trajets
2 - Retour

> 
```


```
Rechercher les trajets
Rentrez la valeur q pour quitter ce menu.
0 - *Ville de depart[]
1 - *Ville d'arrivee[]
2 - *Date de depart (aaaa-mm-jj)[]
3 - Prix minumum[]
4 - Prix maximum[]
```

```
Rentrez la valeur q pour quitter ce menu.
0 - *Ville de depart[Paris]
1 - *Ville d'arrivee[Compiegne]
2 - *Date de depart (aaaa-mm-jj)[2023-02-10]
3 - Prix minumum[]
4 - Prix maximum[]
5 - Rechercher
> 5
---------

  Idx  Depart              Heure depart   Arrivee                 Heure arrivee   Jour         Prix
-----  ------------------  -------------  ----------------------  --------------  ----------  ------
    0  Paris[Gare du Nord]  17:0          Compiegne[Gare Pompidou]  18:0           2023-02-10   10.1

 ----
Souhaitez-vous réserver l'un de ces voyages ?
Entrez la valeur du champ 'Idx' pour choisir un voyage ou q pour quitter
> 
```


```
Consulter les horaires des trains :
- Partant de quelle gare ?
> Gare du Nord
- Dans quelle ville ?
> Paris
- Quand ? aaaa-mm-jj
> 2023-02-10

 Horaire (h:m)      numero de train
---------------  ------------------
13:30                             5
17:0                              6
```

Lorsque l'utilisateur ou l'utilisatrice souhaite rechercher les trajets disponibles, il lui faut indiquer à minima la ville de départ, d'arrivée et la date de départ souhaitée. Puis, si un trajet a été trouvé, l'application lui propose de se connecter pour en réserver un.

III.B.3. Utilisateur/utilisatrice connecté.e

Quand on se connecte à l'application, on doit fournir son numéro de téléphone afin de s'authentifier. Il est possible de réserver un trajet et de consulter les réservations de cette personne.

```
Veuillez vous connecter pour accéder a vos données.
Quel est votre numéro de téléphone ?
Ne rien mettre pour créer un nouvel utilisateur.
Mettre 0 pour revenir au menu précédent.
> 0715151515

Connecté ! Bonjour Noam Seuret
------
0 - Réserver
1 - Consulter les réservations
2 - Déconnexion

>
```

Le menu pour réserver un voyage est similaire à celui de l'utilisateur.rice non connecté.e. Et dans le menu des réservations sont affichées toutes les réservations passées et futures de la personne connectée ainsi que le détail de chaque voyage planifié. Ainsi, on obtient un tableau comme suit :

```
Réservations :
------
  Billet  Date        Départ               Arrivée              Assuré      Prix
       0  2023-05-03  Lyon Part-Dieu[Lyon]  Gare Pere Noel[Nuuk]  False        127
         ------ Détails ------
         0 : Le 2023-05-03(08:00:00) à Lyon(Lyon Part-Dieu) -> Paris(Gare du Nord) le 2023-05-03(11:55:00) | Place : 1
         1 : Le 2023-05-03(13:30:00) à Paris(Gare du Nord) -> Nuuk(Gare Pere Noel) le 2023-05-03(23:12:00) | Place : 23
```

III.B.4. Admin

Enfin, en se connectant à l'application comme admin du système, il devient possible de gérer (i.e. ajouter, modifier, supprimer) les gares, trains, lignes, itinéraires et d'obtenir des statistiques sur le fonctionnement de l'entreprise (taux de remplissage pour chaque voyages, ...).

```
------
0 - Gestion des gares
1 - Gestion des lignes
2 - Gestion des trains
3 - Gestion des itineraires
4 - Statistiques
5 - Retour

>
```

IV- Passage en NoSQL

Pour le passage au NoSQL, on a décidé de faire du relationnel-JSON. En effet, cette solution pour coupler l'imbrication NF^2 avec une base relationnelle nous semblait plus facile à implémenter par rapport aux informations déjà présentes et à la structure de notre base de données .

IV.A - Trajet en attribut JSON de Billet

La transformation :

Avant :

```
CREATE TABLE Billet (
  idBillet INTEGER PRIMARY KEY,
  assurance BOOLEAN NOT NULL,
  acheteur INTEGER NOT NULL,
  typePaiement VARCHAR,
  FOREIGN KEY (acheteur) REFERENCES
Voyageur(idVoyageur),
  FOREIGN KEY (typePaiement)
REFERENCES TypePaiment(intitule)
);
```

```
CREATE TABLE Trajet (
  billet INTEGER,
  numeroTrajet INTEGER,
  numeroPlace SMALLINT NOT NULL,
  voyage INT NOT NULL,
  prix DECIMAL(5,2) NOT NULL,
  heureDepart TIME NOT NULL,
  heureArrivee TIME NOT NULL,
  date DATE NOT NULL, --A tester
  PRIMARY KEY (billet, numeroTrajet),
  CHECK (numeroPlace >= 0 AND numeroTrajet >= 0),
  FOREIGN KEY (billet) REFERENCES Billet(idBillet),
  FOREIGN KEY (voyage) REFERENCES Voyage(id_Voyage)
);
```

Maintenant :

```
CREATE TABLE Billet (
  idBillet INTEGER PRIMARY KEY,
  assurance BOOLEAN NOT NULL,
  acheteur INTEGER NOT NULL,
  typePaiement VARCHAR,
  trajet JSON NOT NULL,
  FOREIGN KEY (acheteur) REFERENCES
Voyageur(idVoyageur),
  FOREIGN KEY (typePaiement) REFERENCES
TypePaiment(intitule)
);
```

```
exemple de trajet
trajet JSON [
{"numeroTrajet" : 1,
 "numeroPlace" : 22,
"prix": 12.5,"
heureDepart": "08:00:00",
"heureArrivee":"11:55:00" ,
"date" : "2023-05-05"
,"voyage":1},
{"numeroTrajet" : 2,
 "numeroPlace": 72,
 "prix": 50,
"heureDepart": "13:30:00",
"heureArrivee":"23:12:00" ,"date" :
"2023-06-08","voyage":5}]
```

La justification :

C'était la transformation la plus évidente : Trajet (composant) et Billet (composite) forment une composition, il est ainsi aisé et intéressant de transformer cette composition en une entrée de type JSON.

IV. B - Planification en attribut JSON de Voyage (et exception)

La transformation :

```
CREATE TABLE Voyage (
  id_Voyage INTEGER,
  heureDepart TIME NOT
NULL,
  train INT NOT NULL,
  ligne INT NOT NULL,
  PRIMARY KEY (id_Voyage),
  FOREIGN KEY (train)
REFERENCES Train(numero),
  FOREIGN KEY (ligne)
REFERENCES Ligne(numero)
);
```

```
CREATE TABLE Plannification (
  idPlanification INT,
  dateDepart DATE NOT NULL,
  dateFin DATE NOT NULL,
  lundi BOOLEAN NOT NULL,
  mardi BOOLEAN NOT NULL,
  mercredi BOOLEAN NOT NULL,
  jeudi BOOLEAN NOT NULL,
  vendredi BOOLEAN NOT NULL,
  samedi BOOLEAN NOT NULL,
  dimanche BOOLEAN NOT NULL,
  PRIMARY KEY (idPlanification),
  CHECK (dateDepart < dateFin),
  CHECK (lundi or mardi or mercredi or
jeudi or vendredi or samedi or dimanche)
);

CREATE TABLE VoyagePlannifie(
    plannification INT REFERENCES
Plannification(idPlanification),
    voyage INT REFERENCES
Voyage(id_Voyage),
    PRIMARY KEY (plannification,
voyage));
```

```
CREATE TABLE
JourException (
  jour DATE,
  ajout BOOLEAN NOT
NULL,
  PRIMARY KEY (jour)
);

CREATE TABLE
ExceptionPlannification(
    jour DATE REFERENCES
JourException(jour),
    plannification INT
REFERENCES
Plannification(idPlanifi
cation),
    PRIMARY KEY(jour,
plannification)

);
```

Maintenant :

```
CREATE TABLE Voyage (
  id_Voyage INTEGER,
  heureDepart TIME NOT NULL,
  train INT NOT NULL,
  ligne INT NOT NULL,
  plannifier JSON NOT NULL,
  PRIMARY KEY (id_Voyage),
  FOREIGN KEY (train) REFERENCES
Train(numero),
  FOREIGN KEY (ligne) REFERENCES
Ligne(numero)
);
```

```
exemple :  de plannifier
pannifier JSON
    {
            "dateDepart" : "2023-01-01",
            "dateFin": "2023-06-01",
            "lundi": true,
            "mardi": true,
            "mercredi": true,
               "jeudi": true,
               "vendredi": true,nom
               "samedi": true,
               "dimanche": true,
            "exception": [ { "jour":
"2023-05-01", "ajout": false }]
    },
    {
          "dateDepart" : "2023-06-02",
          "dateFin": "2023-12-31",
          "lundi": true,
          "mardi": true,
          "mercredi": true,
        "jeudi": true,
        "vendredi": true,
        "samedi": true,
          "dimanche": true,
          "exception": [ { "jour":
"2023-07-03", "ajout": false }]
        }
]
```

La justification :

Pour cette transformation, c'est les requêtes pour obtenir les voyages disponibles qui nous ont mis sur la piste :

```
-- Un utilisateur veut savoir quels voyages sont disponibles pour --faire
Paris-Compiègne le dimanche 05\03\2023
SELECT vg.voyage,vg.heureDepart,vd.heurearrivee
FROM voyageDessert vg
INNER JOIN voyageDessert vd ON vg.voyage=vd.voyage
INNER JOIN VoyagePlannifie vp ON vg.voyage=vp.voyage
INNER JOIN Plannification p ON vp.plannification=p.idPlanification
INNER JOIN ExceptionPlannification ep  ON ep.plannification=p.idPlanification
INNER JOIN JourException je ON ep.jour =je.jour
WHERE vg.villeGare='Paris' AND vd.villeGare='Compiegne' AND
p.dateDepart<='2023-03-05' AND p.dateFin>='2023-03-05' AND (p.dimanche=TRUE OR
(je.jour='2023-03-05' and je.ajout=TRUE)) AND NOT (je.jour='2023-03-05' and
je.ajout=FALSE);
```

En effet, on observe de nombreuses jointures pour l'obtention d'une information qui sera demandée de manière très récurrente. Le NoSQL semble être un véritable allié pour rendre l'accès à l'information plus facile, bien que l'on gagne en redondances car la table sera moins normalisée. On a ainsi affaire dorénavant à une double imbrication : exception –<> Planification –<> Voyage.

IV.C - Hôtel et Transport en attribut JSON de Gare

La transformation :

avant :

```
CREATE TABLE Gare (
  nom VARCHAR,
  ville VARCHAR,
  numeroVoie SMALLINT NOT NULL,
  nomRue VARCHAR NOT NULL,
  codePostal CHAR(5) NOT NULL,
  zoneHorraire SMALLINT NOT NULL CHECK (zoneHorraire >= -12 AND zoneHorraire <= 12),
  PRIMARY KEY (nom, ville)
);
```

```
CREATE TABLE Hotel (
  id_Hotel INT,
  nom VARCHAR NOT NULL,
  numeroVoie SMALLINT NOT NULL,
  nomRue VARCHAR NOT NULL,
  codePostal CHAR(5) NOT NULL,
  ville VARCHAR NOT NULL,
  PRIMARY KEY (id_Hotel)
);

CREATE TABLE HotelProcheDeGare (
    hotel INT REFERENCES Hotel(id_Hotel),
    nom_gare VARCHAR,
    ville_gare VARCHAR,
    FOREIGN KEY (nom_gare, ville_gare) REFERENCES Gare(nom, ville),
    PRIMARY KEY(hotel, nom_gare, ville_gare)
);
```

```
CREATE TABLE Transport (
  id_Transpot INT,
  type TypeTransport NOT NULL,
  ville VARCHAR NOT NULL,
  PRIMARY KEY (id_Transpot)
);

CREATE TABLE TransportProcheDeGare(
    transport INT REFERENCES Transport(id_Transpot),
    nom_gare VARCHAR,
    ville_gare VARCHAR,
    FOREIGN KEY (nom_gare, ville_gare) REFERENCES Gare(nom, ville),
    PRIMARY KEY(nom_gare, ville_gare, transport)
);
```

Après :

```
CREATE TABLE Gare (
  nom VARCHAR,
  ville VARCHAR,
  adresse JSON,
  zoneHorraire SMALLINT NOT NULL CHECK
(zoneHorraire >= -12 AND zoneHorraire <=
12),
  hotel JSON,
  transport JSON,
  PRIMARY KEY (nom, ville)
);
```

```
exemple d'hôtel et de transport :
hôtel JSON
    [{
    "id_Hotel": 5,
    "nom": "Chez le marchand de sable",
     "ville": "Valence",
    "adresse":{
    "numeroVoie": 89,
    "nomRue": "Rue des mysteres",
    "codePostal": 26000
    }
    }]
 transport JSON
[{ "type":"BUS" }, { "type":"TAXI" }, {
"type":"Metro" }]
```

La justification :

Cette transformation est la plus arbitraire, elle n'a pas grand intérêt parce qu'elle a autant d'avantages que d'inconvénients. D'une part, on enlève les jointures pour l'accès à l'information, mais de l'autre côté, on crée de la redondance. Ainsi cette transformation a plus été réalisée dans l'intention de montrer notre capacité à maîtriser le NoSQL qu'autre chose.

Bonus : les adresses ont aussi été transformées en JSON.

Ces modifications ont ainsi dû être reportées dans les insertions de valeurs, et dans les requêtes.

Conclusion

Le projet de gestion des trains pour la société de chemins de fer a été une expérience enrichissante, mettant en évidence des aspects tels que la gestion de l'imprévu, la rigueur des rendus hebdomadaires et la nature fastidieuse du travail de création et d'alimentation de Base de Donnée.

Tout d'abord, au début du projet, nous avons été confrontés à des changements de groupe, ce qui a nécessité de reformer les équipes de travail. Cela nous a permis de développer notre adaptabilité et notre capacité à collaborer efficacement dans différentes configurations.

L'approche rigoureuse avec des rendus hebdomadaires nous a incité à maintenir un rythme de travail soutenu et à respecter les délais impartis. Cela a renforcé notre discipline et notre capacité à gérer notre temps de manière efficace. Bien que le travail puisse parfois sembler fastidieux, nous avons appris à persévérer et à surmonter les défis qui se présentaient à nous.

L'un des aspects les plus importants et déterminants de ce projet a été la phase de conception. En réalisant ce projet, on a compris toute l'importance de passer du temps à la création d'un Modèle Conceptuel de Données (MCD), d'un Modèle Logique de Données (MLD) de qualité. En effet, c'est de ces derniers que découle la facilité, ou non, de leur mise en œuvre dans une base de données relationnelle (PSQL).

De plus, ce projet nous a offert l'occasion d'apprendre et de maîtriser le langage SQL, qui joue un rôle crucial dans la manipulation des données dans une base de données relationnelle. Nous avons acquis une compréhension pratique des requêtes SQL, ce qui nous permettra d'appliquer ces connaissances dans des projets futurs.

En conclusion, le projet de gestion des trains a été une expérience exigeante mais gratifiante. Il nous a permis de développer nos compétences en matière de gestion de projet, de collaboration d'équipe, de conception de bases de données relationnelles et non relationnelles, ainsi que d'apprentissage du langage SQL. Ce projet nous a préparés à relever de nouveaux défis dans le domaine du développement logiciel et nous a dotés d'une base solide pour nos futures réalisations.