Projet de Base de Données Relationnelles

Socrimoft Railway's

Clolot Ludovic, Duban Louis, Monsterleet Baptiste, Reynier Dorian
Enseignant référent : Nadia Abchiche

Polytech Nice Sophia SI3 2024-2025

Table des matières

I. Description du projet

Séparation du travail

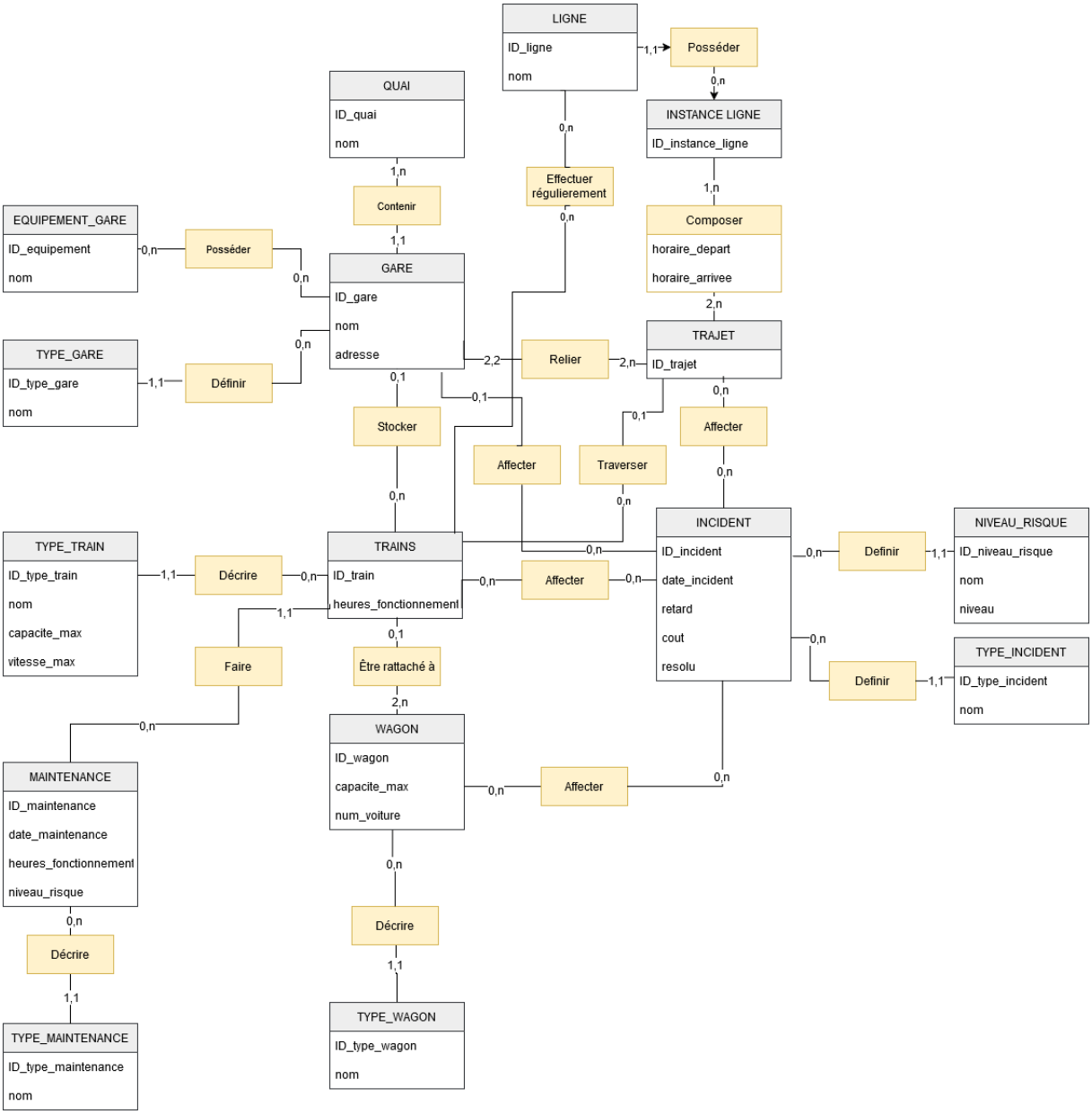
Lien vers GitHub

Socrimoft Railway's est une base de données ayant pour but de gérer un réseau ferroviaire national ou international.
Pour cela, les informations des trains, trajets, lignes, gares, horaires ainsi que leur état et équipement sont stockées dans différentes tables et interagissent entre elles pour permettre à un utilisateur d'obtenir de nombreuses informations,
mais également de lui donner accès à plusieurs fonctionnalités, telles qu'un ajout simplifié de lignes, une fonction d'optimisation des trajets/horaires,
une gestion des maintenances ainsi que des rappels pour les effectuer, des fonctions de gestion de gares et de monitoring des flux de passagers, des potentielles saturations, et un système de gestion des pannes et de l'équipement pour éviter une immobilisation du trafic.

II. Schéma de la base de données

Modèle Entité Association

Lien vers l'image sur GitHub



Explications

Chaque élément de chaque table est identifié par son identifiant.

Trains :

- Les trains sont définis par un type de train
- Les données comme la vitesse maximum et la capacité maximum de wagon est défini dans le type, car tous les trains du même type auront les mêmes attributs et cela permet d'éviter de la duplication de données.
- Un train peut avoir un incident
- Un train peut être rattaché aucun wagon ou plusieurs
- Un train peut avoir des maintenances
- Un train peut être stocké dans une gare
- Un train peut traverser un trajet
- Un train peut avoir une ligne habituelle

Maintenances :

- Une maintenance possède une date de réalisation
- Une maintenance stocke les heures de fonctionnement du train au moment de la maintenance
- Une maintenance stocke le niveau de risque du train au moment de la maintenance (somme du niveau de risque des incidents du train

Wagon :

- Nous avons choisi de séparer les wagons des trains car des incidents peuvent survenir sur un wagon et non un train (exemple : panne de climatisation)
- Les wagon sont définis par un type de wagon qui permet de classifier les différents wagon (exemple : wagon restaurant, wagon passagers, ...)
- Le wagon possède un nombre de voyageur max (qui peut être différent pour chaque wagon)
- Un wagon peut être lié à un train
- Le wagon possède un numéro de voiture quand il est lié à un train qui permet de le situer dans le train
- Un wagon peut avoir un incident

Trajet :

- Un trajet relie deux gares, il possède une gare de départ et une gare d'arrivée
- Il peut y avoir un incident sur un trajet

Lignes :

- Une ligne possède un nom et peut avoir aucune ou plusieurs instances de ligne
- Les instances des lignes permettent de créer un itinéraire d'une gare à l'autre en suivant plusieurs trajets à des horaires précises qui se suivent, on stocke les horaires de départ et d'arrivée pour chaque trajet de l'instance de ligne

Gare :

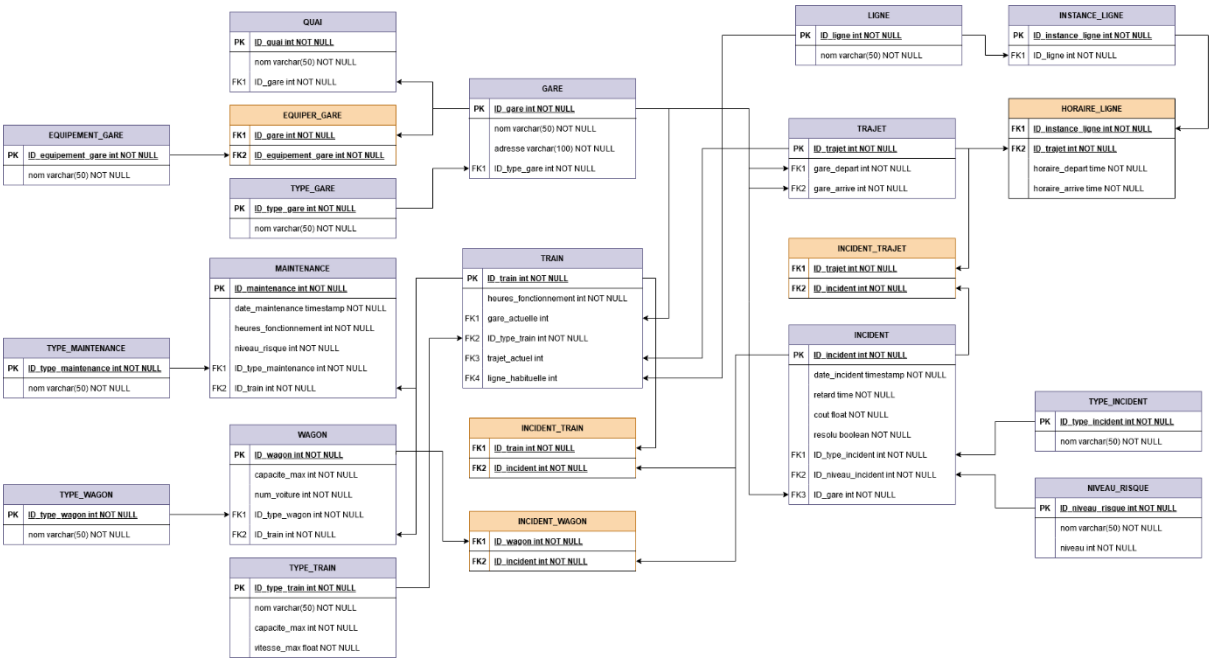
- Une gare est définie par un type de gare (exemple : gare passager, gare industrielle, ...)
- Une gare possède un nom et une adresse
- Une gare possède des équipements
- Une gare possède au moins un quai et peut en avoir plus
- Une gare peut avoir un incident

Incidents :

- Un incident est défini par un type d'incident (exemple : roues endommagées, phares en panne, ...)
- Un incident possède un niveau de risque qui permet d'indiquer l'importance de l'incident
- Un incident possède une date d'incident qui indique quand l'incident a eu lieu
- Un incident possède une valeur de retard qui indique le retard causé par l'incident
- Un incident possède un cout qui correspond au cout de résolution de l'incident
- Un incident possède une valeur booléenne qui indique si l'incident est résolu ou non

Modèle Relationnel

Lien vers l'image sur GitHub



Transition du modèle EA à Relationnel

Afin de faire la transition du modèle entité association au modèle relationnel, nous avons suivi deux conditions :

- S'il y a une relation n-n, alors une nouvelle table est créée avec un nom rappelant les deux tables (exemple : **incident_trajet**). Cette nouvelle table utilise deux clés étrangères qui sont les identifiants des deux tables de la relation et ces clés étrangères vont former la clé primaire de la nouvelle table.
- S'il y a une relation n-x, alors une clé étrangère est rajoutée dans la relation du côté du « n » (exemple : une **gare** ne peut définir qu'un seul **type_gare** et un **type_gare** peut définir 0 à n **gare**, dans le schéma, le n est du côté de **gare** donc **gare** va obtenir la clé étrangère **id_type_gare**)

En suivant ces deux conditions, nous pouvons obtenir toutes les tables que nous devons implémenter dans la base de données.
Il ne reste plus qu'à définir les types de chaque attribut de chaque table qui sont souvent très logique à implémenter mais certaines exceptions ont été nécessaires :

- Les attributs **heures_maintenance** sont de type **int** car **time** ne peut pas dépasser les 24 heures
- Pour les dates (exemple : **date_maintenance**), nous utilisons le type **timestamp** car c'est celui utilisé par pgsql pour les dates

III. Requêtes sur la base de données

1) Quels trajets reliant deux gares données minimisent les correspondances ou les temps d'attente ?

// Ludovic Clolot

Lien vers le fichier sur GitHub

Explications

Objectif : Trouver les trajets optimaux entre deux gares en minimisant soit le nombre de correspondances, soit le temps d'attente total.

Approche :

Modélisation récursive du réseau :

Utilisation d'une CTE récursive pour explorer tous les chemins possibles entre la gare de départ et la gare d'arrivée.

Chaque itération ajoute un segment de trajet valide (même ligne ou correspondance).

Optimisation multicritère :

Calcul simultané du nombre de correspondances (changements de ligne) et du temps d'attente cumulé entre les trajets.

Priorisation des résultats :

Tri primaire par nombre de correspondances (minimiser les changements).

Tri secondaire par temps d'attente total (minimiser l'attente).

Gestion des contraintes :

Exclusion des cycles via NOT trajet_id = ANY(chemin_trajets).

Filtrage des horaires incompatibles (horaire_depart >= horaire_arrive_prev).

Choix techniques :

CTE récursive plutôt que pgRouting pour :

Une meilleure intégration avec les contraintes temporelles.

Un contrôle fin des critères d'optimisation.

Résultats :

Retourne une liste de trajets classés par optimalité.

Colonnes :

chemin_trajets : Liste ordonnée des identifiants de trajets.

nb_correspondances : Nombre de changements de ligne.

temps_attente_total : Temps cumulé entre les correspondances.

2) Quels trains nécessitent une maintenance basée sur leurs heures cumulées de trajet ou des incidents signalés ?

// Baptiste Monsterleet

Requête SQL

Lien vers le fichier sur GitHub

Explications

La vue renvoie la liste de tous les trains qui remplissent au moins l'un des deux conditions :

I) Le train a cumulé un niveau de risque supérieur ou égal à 3 depuis sa dernière maintenance

II) Le train à cumulé 3000 heures de trajet ou plus depuis sa dernière maintenance

Si le train n'a jamais eu de maintenance auparavant, on regarde si ses données actuelles remplissent les conditions précédentes

Nous utilisons une vue car cela permet de facilement obtenir la liste des trains qui ont besoin d'une maintenance. De cette façon, on peut ensuite librement planifier les maintenances des trains.

L'utilisation de triggers pour la planification automatique de maintenance des trains a été envisagée mais cette méthode n'a pas été retenue car elle nous priverait de la liberté de planification des maintenances.

La vue possède 3 colonnes :

1) **id_train** : l'identifiant du train qui nécessite une maintenance

2) **heures_depuis_derniere_maintenance** : nombre d'heures de trajet du train depuis sa dernière maintenance (ou depuis sa mise en service si il n'a jamais de maintenance)
3) **risque_depuis_derniere_maintenance** : la somme des niveau de risque des incidents qui lui sont arrivées depuis sa dernière maintenance (ou depuis sa mise en service si il n'a jamais de maintenance)

Dans le *SELECT*, nous récupérons **id_train** depuis la table **train**, puis nous récupérons **heures_fonctionnement** de ce train auquel on soustrait soit **heures_fonctionnement** stocké dans la dernière maintenance du train concerné, soit à 0 si le train n'a jamais eu de maintenance (*COALESCE*).

Nous appliquons une logique similaire pour le niveau de risque. Nous récupérons d'abord la somme (*SUM*) des niveaux de risque de tous les incidents dans lesquels le train est impliqué (ou 0 si aucun : *COALESCE*). Nous soustrayons ensuite à cette valeur soit **niveau_risque** stocké dans la dernière maintenance du train concerné, soit 0 si le train n'a jamais eu de maintenance (*COALESCE*).

Nous faisons ensuite des jointures avec les tables **incident_train**, **incident** et **niveau_risque** afin de récupérer tous les incidents de chaque train et leur niveau de risque. Nous effectuons une sous requête pour la dernière jointure qui nous permet de récupérer les maintenances les plus récentes de chaque train.

Pour chaque jointure dans cette requête nous utilisons des *LEFT JOIN* afin de pouvoir conserver dans la vue les trains qui n'aurait eu aucune maintenance ou aucun incident auparavant.

Enfin, on regroupe sur **id_train** et **heures_fonctionnement** de **train**, **niveau_risque** et **heures_fonctionnement** des maintenances les plus récentes. *HAVING* nous permet de finaliser la requête en rajoutant les conditions mentionnées au début.

3) Quelles sont les gares ayant une saturation de trafic selon les heures de pointe ?

// Baptiste Monsterleet

Requête SQL

Lien vers le fichier sur GitHub

Explications

Ici, nous utilisons 4 vues différentes pour atteindre notre objectif. Nous avons utilisé les vues afin de simplifier et faciliter la rédaction de la requête finale. La requête finale est également une vue car elle permet d'avoir une vue globale sur la surcharge de toutes les

gares et facilite la potentielle réorganisation des horaires pour éviter les saturations de trafic.

Nous avions envisagé l'utilisation de fonctions pour la requête finale où nous aurions pu choisir la gare que l'on souhaite analyser, mais cela empêche la possibilité d'avoir une vue d'ensemble de toutes les surcharge. De plus, il est très facile de sélectionner uniquement les gares qui nous intéressent dans la vue en utilisant *WHERE*.

Les vues **depart_gare** et **arrive_gare** sont très similaires, la première affiche l'ensemble des horaires de départs des trains depuis chaque gare et le deuxième fait pareil mais pour les arrivées de trains.

Pour la vue **gare_surcharge_prevue_tout_horaire**, on calcule l'indice de surcharge prévu pour une horaire donnée pour chaque gare.

Plus l'indice de surcharge est élevé, plus la gare est sensée être surchargé à l'horaire indiqué. L'indice indique la surcharge théorique de la gare, pas un nombre de train précis.

L'indice est calculé ligne par ligne, en suivant l'ordre croissant des horaires et est indépendant pour chaque gare.

Pour chaque train qui part de la gare, l'indice est réduit de 1 et pour chaque train qui arrive dans la gare, l'indice est augmenté de 1

Cela signifie que l'indice peut être négatif, c'est un comportement normal et prévu étant donné que l'indice ne représente pas un nombre réel de train dans une gare.

Les deux colonnes **horaire_debut** et **horaire_fin** sont nécessaires pour savoir si l'horaire est celle d'un train qui part où qui arrive. Il peut y avoir le même horaire dans deux colonnes sur la même ligne si deux trains partent et arrivent au même moment.

Nous obtenons les deux colonnes grâce à une jointure de type *FULL OUTER JOIN* et aux deux vues précédentes.

L'utilisation de *SUM* et de *CASE* permet de calculer l'indice de surcharge. Avec *SUM*, nous utilisons *OVER* et *PARTITION BY* sur **id_gare** afin que les indices de surcharges de chaque puissent être calculé indépendamment.

Dans *OVER*, nous utilisons *CASE* dans le *ORDER BY* afin de pouvoir trier sur les deux colonnes simultanément, si nous ne trions pas de cette façon, les deux colonnes serait triées l'une après l'autre et le calcul de l'indice aurait été faussé.

La vue **gare_surcharge_prevue** est celle qui répond à la question. Elle affiche tous les intervalles de temps ou l'indice de surcharge est supérieur ou égal à 2 pour chaque gare.

Dans cette vue, nous utilisons d'abord *WITH* dans lequel on utilise la vue précédente pour créer des intervalles d'horaires. Ces intervalles possèdent un indice de surcharge comme les horaires de la vue précédente.

Pour calculer ces intervalles, nous parcourons ligne par ligne le résultat de la vue précédente :

- L'horaire de début de l'intervalle est l'horaire de la ligne actuelle
- L'horaire de fin de l'intervalle est l'horaire stocké sur la ligne suivante (utilisation de *LEAD* pour récupérer le résultat de la ligne suivante)
- L'indice de surcharge est celui de la ligne actuelle

Nous utilisons *CASE* pour vérifier que la ligne suivante appartient à la même gare que la ligne actuelle. Si ce n'est pas le cas, on écrit *NULL*.

L'utilisation de « *ORDER BY(SELECT NULL)* » permet de ne pas ordonner car l'ordre à déjà été fait dans la vue précédente.

COALESCE est utilisé pour récupérer les horaires de début et de fin car dans la vue précédente, l'horaire peut être dans une seule des deux colonnes.

Après la création du WITH, nous sélectionnons uniquement les intervalles qui ont un horaire de départ et d'arrivée différentes (les intervalles où les deux horaires sont identiques ne sont pas utiles) et les intervalles qui ont un indice de surcharge supérieur ou égal à 2.

4) Comment intégrer une nouvelle ligne ferroviaire dans le réseau existant tout en optimisant les correspondances ?

//Dorian

Pour cette question, nous allons utiliser une suite de fonctions qui peuvent être réutilisées dans d'autres parties du code. Ces fonctions permettent de gérer et d'ajouter des informations relatives aux lignes, trajets, horaires et de trouver des trajets optimaux. Les fonctions sont les suivantes :

ajouter_ligne : Cette fonction permet de créer une nouvelle ligne de train en insérant un nouvel enregistrement dans la table des lignes et en appelant la fonction pour ajouter une instance de ligne associée.

- **ajouter_INSTANCE_LIGNE** : Elle permet de créer une nouvelle instance de ligne associée à une ligne spécifique en ajoutant un enregistrement dans la table des instances de lignes.

- **ajouter_Trajet** : Cette fonction ajoute de nouveaux trajets entre deux gares, en insérant les trajets dans la table des trajets dans les deux sens (gare de départ et gare d'arrivée inversées).

1. **ajouter_horaires** : Cette fonction permet de créer des horaires pour les trajets, en ajoutant des horaires de départ et d'arrivée dans la table des horaires pour chaque trajet.

    - **trouver_chemin_optimal** : Elle utilise une requête récursive pour trouver le chemin optimal entre deux gares, en considérant le nombre de correspondances et le temps d'attente total.

- **main_ajouter_Ligne** : Fonction principale qui regroupe toutes les autres fonctions. Elle ajoute une nouvelle ligne, les trajets associés, les horaires, et retourne également le chemin optimal entre les gares de départ et d'arrivée.

La plupart de ces fonctions retournent des résultats vides (de type `VOID`), car elles se contentent d'effectuer des actions telles que l'insertion de données ou la mise à jour des tables.

5) Quels trajets alternatifs sont disponibles en cas d'incident sur une ligne spécifique ?

// Ludovic Clolot

Lien vers le fichier sur GitHub

Explications

Objectif : Identifier des itinéraires de remplacement lorsqu'une ligne est partiellement ou totalement bloquée.

Approche :

Identification des trajets bloqués :

Jointure entre les tables INCIDENT_TRAJET et TRAJET pour lister les segments indisponibles.

Recherche récursive de contournement :

Exclusion explicite des trajets bloqués via NOT IN (SELECT id_trajet FROM trajets_incidentes).

Calcul des correspondances et temps d'attente comme pour la question 1.

Garantie de continuité :

Vérification de la cohérence temporelle entre les segments (horaire_depart >= horaire_arrive_prev).

Utilisation de LEFT JOIN pour conserver les trains sans incident.

Choix techniques :

Réutilisation de la CTE récursive de la question 1 avec filtrage supplémentaire.

Pénalisation implicite des trajets alternatifs :

Un chemin avec plus de correspondances mais moins d'attente peut être préféré selon le contexte.

Résultats :

Liste des trajets évitant la ligne incidentée.

Colonnes :

Trajet alternatif : Séquence des trajets empruntés.

Nb correspondances : Complexité du parcours.

Temps attente : Délai supplémentaire induit.

6) Quels trains peuvent être réaffectés pour couvrir une panne sur une autre ligne ? - Louis DUBAN

Requête SQL

Lien vers le fichier sur GitHub

Explications

Objectif : Cette requête permet d'identifier quels trains peuvent être réaffectés pour remplacer un train en panne sur une autre ligne.

Il n'y a qu'une seule requête, car elle est construite avec des **CTE (`WITH`)** qui renvoie un tuple (id du train en panne, id du train disponible au remplacement, nom de la gare ou se situe le train de remplacement).

Étapes de la requête

1) Identifier les trains en panne (`Train_en_panne`)
    a. Sélectionne les trains qui sont actuellement en circulation (ont un `trajet_actuel` défini).
    b. Vérifie qu'ils ont un incident critique (niveau de risque ≥ 2) et non résolu (`resolu = FALSE`).

2) Déterminer les gares impactées (`Gares_Impactees`)
    a. Récupère les gares de départ et d'arrivée des trajets des trains en panne.
    b. Cela permet de savoir où il faudrait un train de remplacement.

3) Identifier les trains disponibles (`Trains_Disponibles`)
    a. Sélectionne les trains situés dans les gares impactées.
    b. Vérifie que ces trains ne subissent aucun incident non résolu, pour s'assurer qu'ils peuvent être réaffectés.

4) Associer les trains en panne aux trains disponibles
    a. Associe chaque train en panne à un train disponible uniquement si ce dernier est dans une gare du trajet affecté.
    b. Affiche l'ID du train en panne, l'ID du train de remplacement, et le nom de la gare où se trouve le train de remplacement.
    c. Trie le résultat par `id_train_panne croissant`.

7) Quels incidents ont le plus grand impact sur la ponctualité globale du réseau ferroviaire ? - Louis DUBAN

Requête SQL

Lien vers le fichier sur GitHub

Explications

Objectif : Ces requêtes permettent d'identifier quels incidents ont le plus grand impact sur la ponctualité du réseau ferroviaire.

Il y a **trois requêtes**, car elles analysent le problème sous différents angles afin d'obtenir une vue d'ensemble.

Approche 1 : Classement des types d'incidents par retard total

Cette requête permet d'identifier les types d'incidents qui génèrent le plus de retard en minutes sur l'ensemble du réseau. Renvoie un tuple (Type d'incident, retard total générés par ce type d'incident en minutes).

Étapes de la requête

a. Sélectionne les incidents en fonction de leur type.

b. Convertit la durée des retards en minutes.

c. Calcule la somme des retards pour chaque type d'incident.

d. Trie les résultats du plus impactant au moins impactant.

Approche 2 : Impact des incidents par type d'infrastructure

Cette requête permet d'analyser l'impact des incidents en fonction du type d'infrastructure affectée (train, wagon, trajet ou gare). Renvoie 4 tuple (catégorie, retard total générés par les incidents sur cette catégorie d'infrastructure).

Étapes de la requête

a. Sélectionne les incidents selon l'infrastructure concernée (train, wagon, trajet, gare)
b. Convertit la durée des retards en minutes.

c. Calcule la somme des retards pour chaque type d'infrastructure.
d. Trie les résultats du plus impactant au moins impactant.

Approche 3 : Incidents qui affectent le plus de trains

Cette requête identifie les types d'incidents qui perturbent le plus grand nombre de trains. Renvoie un tuple (Type d'incident, nombres de train affectés par ce type d'incident).

Étapes de la requête

a. Sélectionne les incidents en fonction de leur type.

b. Compte le nombre de trains distincts affectés par chaque type d'incident.

c. Regroupe les résultats par type d'incident.

d. Trie les résultats du plus impactant au moins impactant.

IV. Description du jeu de données

1) Quels trajets reliant deux gares données minimisent les correspondances ou les temps d'attente ?

// Ludovic Clolot

Lien vers le jeu de données utilisé

Objectif : Tester tous les cas d'optimisation entre correspondances et temps d'attente.

Scénarios couverts :
Trajet direct sans correspondance

Trajet avec 1 correspondance mais temps d'attente minimal

Trajet avec 2 correspondances mais plus rapide

Trajet circulaire (test de cycle)

Trajet avec horaires multi-jours

2) Quels trains nécessitent une maintenance basée sur leurs heures cumulées de trajet ou des incidents signalés ?

// Baptiste Monsterleet

Lien vers le jeu de données utilisé

Dans le jeu de données, nous créons un ensemble de trains, de maintenances et d'incidents afin de tester la sortie de la requête.

Ce jeu de données nous permet de tester les cas suivants :

1) Un train qui n'a pas besoin de maintenance et qui n'en a jamais eu (n'apparait pas dans le résultat de la requête)
2) Un train qui n'a jamais eu de maintenance mais qui a cumulé plus de 3000 heures de trajet
3) Un train qui n'a jamais eu de maintenance mais qui a cumulé un niveau de risque égal à 3
4) Un train qui a déjà eu une maintenance à cause des heures de trajet et qui n'a pas besoin de nouvelle maintenance (n'apparait pas dans le résultat de la requête)

5) Un train qui a déjà eu une maintenance à cause de niveau de risque et qui n'a pas besoin de nouvelle maintenance (n'apparait pas dans le résultat de la requête)
6) Un train qui a déjà eu une maintenance à cause des heures de trajet et qui a besoin d'une nouvelle maintenance à cause des heures de trajet
7) Un train qui a déjà eu une maintenance à cause de niveau de risque et qui a besoin d'une nouvelle maintenance à cause du niveau de risque
8) Un train qui a déjà eu deux maintenances : une pour les heures de trajet et une autre pour le niveau de risque. Elle a besoin d'une nouvelle maintenance à cause des heures de trajet (Permet de tester que l'on récupère bien la maintenance la plus récente)

Tous les tests ont réussi ce qui permet d'assurer le fonctionnement de la requête.

3) Quelles sont les gares ayant une saturation de trafic selon les heures de pointe ?

// Baptiste Monsterleet

Lien vers le jeu de données utilisé

Dans ce jeu de données, nous créons un ensemble de lignes, de gares et de trajets afin de pouvoir tester la requête.

Les horaires des lignes ont été pensés de façon qu'il y ait deux intervalles de temps dans la gare 2 où l'indice de surcharge est supérieur à 2.

Dans le résultat de la requête, on peut observer ces deux intervalles pour la gare 2 :

- De 6h55 à 7h05
- De 7h30 à 7h35

Ce test nous permet de valider le fait qu'une gare peut avoir plusieurs intervalles de temps où il y a une surcharge et que les intervalles de surcharges peuvent être de longueur différente.

4) Comment intégrer une nouvelle ligne ferroviaire dans le réseau existant tout en optimisant les correspondances ?

//Dorian

Aucun jeu de donnée n'était nécessaire pour tester la création de lignes. Le fichier q4data.sql contient tout de même des données car elles permettent une visualisation plus agréable des changements.

5) Quels trajets alternatifs sont disponibles en cas d'incident sur une ligne spécifique ?

// Ludovic Clolot

Lien vers le jeu de données utilisé

Objectif : Tester la résilience du réseau avec différents types d'incidents.

Scénarios critiques :

Incident sur le trajet direct unique

Incident sur une ligne complète

Incident multi-trajets sur différents axes

Aucun trajet alternatif possible

6) Quels trains peuvent être réaffectés pour couvrir une panne sur une autre ligne ? - Louis DUBAN

Insertion SQL

Lien vers le fichier sur GitHub

Données permettant de tester la requête q6requete.sql de manière unitaire en testant différentes combinaisons possibles d'incidents sur les trains en panne et disponible.

Par exemple : - Associer un incident non résolu à un train qui devrait être disponible au remplacement ou au contraire vérifier qu'un train possédant que des incidents résolus ne soit pas considéré comme en panne.

- De plus on rattache des trains à des gares qui ne sont pas concernés par des pannes de trains proche pour vérifier que ces trains ne figurent pas dans la partie "Train_Disponible" de la requête.

7) Quels incidents ont le plus grand impact sur la ponctualité globale du réseau ferroviaire ? - Louis DUBAN

Insertion SQL

Lien vers le fichier sur GitHub

Données permettant de tester les requêtes q7requete.sql de manière unitaire en testant le retard causer par différents types d'incidents sur différentes structures.

Par exemple :
- Une panne motrice d'une locomotive ou bien un problème de caténaire affectant un trajet complet.
- On test des pannes sur plusieurs infrastructures (trains, gare, wagon, trajet).

V. Analyse critique

Dans l'ensemble, notre équipe a su bien répartir le travail, ce qui nous a permis d'avancer efficacement sans rencontrer de véritables difficultés bloquantes. Chaque membre a pris en charge une partie des tâches, et nous avons pu progresser de manière organisée. Cependant, il est important de souligner que Baptiste a pris une charge de travail plus importante que les autres, notamment en codant la base de données en SQL et en finalisant les modèles.

Concernant les défis rencontrés, nous n'avons pas eu de problème majeur en dehors d'un léger manque de coordination sur certaines parties, notamment lors de l'exécution des requêtes SQL. Par exemple, nous aurions pu mieux synchroniser nos tests pour éviter de devoir clear entre nos insertions de données. Une solution simple aurait été d'attribuer des ID fixes à chacun, ce qui aurait permis d'éviter ces ajustements répétitifs.

Si c'était à refaire, nous mettrions davantage l'accent sur la communication au sein de l'équipe, afin de mieux coordonner nos requêtes, nos tests et d'éviter de dupliquer nos données.

Malgré ce point d'amélioration, le projet s'est bien déroulé grâce à une bonne répartition initiale des tâches et à l'investissement de chacun.

VI. Vidéo de présentation du projet

Lien vers la vidéo sur Google Drive