Le sujet comprend deux parties interdépendantes. Lisez bien l'intégralité du sujet avant de commencer. Donnez les réponses sur le sujet que vous joindrez à votre copie.

1 Transports en commun urbain : contraintes et requêtes

Dans tous les exercices suivants, on fixe une base de données utilisée par un service de transport en commun urbain. La base de données respecte le schéma ci-dessous, où les clefs primaires sont soulignées et les clefs étrangères listées à la suite.

`station(idS, nameS, lattitude, longitude, prm)`
`line(idL, firstS, lastS, auto)`
`route(idR, line, sourceS, destS, deptime, arrtime)`
`distance(destS, sourceS, meters)`

Clefs étrangères : `firstS`, `lastS` dans `line`, `destS`, `sourceS` dans `route` et dans `distance` font référence à `idS` dans `station` ; `line` dans `route` fait référence à `idL` dans `line`.

Ce service comporte plusieurs lignes de métro (tuples de la table `line`). Certaines lignes sont automatiques, c'est à dire que les trains circulent sans conducteur, d'autres non (attribut de type Booléen `auto`).

Certaines des stations (tuples de la table `station`) desservies par ces lignes sont également stations de départ (attribut `firstS` dans la table `line`) ou bien terminus (attribut `lastS` dans la table `line`) d'une ou plusieurs ligne(s). Certaines stations sont accessibles aux personnes à mobilité réduite, d'autres non (attribut de type Booléen `prm`).

Les itinéraires possibles peuvent être reconstruits grâce à la table `route`. On suppose que les horaires de départ et d'arrivée des trains sont identiques quel que soit le jour de l'année et que chaque métro s'arrête à toutes les stations de la ligne qu'il parcourt. Dans la table `route`, pour une ligne `line` donnée les attributs `destS, sourceS` représentent un couple de stations adjacentes sur cette ligne, avec `deptime` un horaire de départ depuis `destS` et `arrtime` l'horaire d'arrivée à `arrS` correspondant. Par exemple sur la ligne 14 le métro qui part de Châtelet à 5h34 arrive à Gare de Lyon à 5h37. La table `distance` représente la distance séparant deux stations `destS` et `sourceS`. Par exemple la distance séparant Châtelet et Gare de Lyon est de 2487 mètres.

Notez que ce schéma n'est pas parfait. Il vous sera demandé dans la deuxième partie du sujet de l'améliorer.

Exercice 1 : Lecture du schéma relationnel

Pour chacune des questions suivantes, écrivez `autorisé` si le schéma permet les situations décrites et `interdit` sinon.

1) Une ligne a pour gare de départ son terminus.

Solution: oui

2) Pour une même ligne, un même horaire de départ et une même station de départ, la base de données contient deux horaires différents d'arrivée à la même station.

Solution: oui

3) Il y a une station sans nom.

Solution: oui

Exercice 2 : création des tables

Pour chacune des questions suivantes, dites quels choix faire (contraintes, types de données, etc.) à la création des tables pour obtenir les comportements voulus.

La syntaxe SQL précise n'est pas obligatoire : indiquez seulement la nature du choix, et les tables et attributs concernés.

1) Deux stations différentes ont des longitudes et lattitudes différentes.

Solution: Ajouter une contrainte d'unicité sur ce couple d'attributs

2) L'accessibilité des stations aux personnes à mobilité réduite est forcément renseignée.

Solution: Ajouter une contrainte de non nullité sur `prm`

3) Si une ligne est rebaptisée (par exemple, si on décide de rebaptiser ligne 42 la ligne 14), cette modification est automatiquement répercutée dans l'ensemble de la base de données.

Solution: Supprimer la contrainte de clef étrangère et la récréer en spécifiant ON UPDATE CASCADE

4) Il est impossible de se retrouver dans le type de situation suivante. La distance entre Châtelet et Gare de Lyon est de 2487 mètres. En revanche la distance entre Gare de Lyon et Châtelet est de 487 mètres.

Solution: La contrainte de vérification suivante suffit à empêcher cette contrainte, puisque seule la distance entre Châtelet et Gare de Lyon sera explicitement représentée ("Gare de Lyon" n'étant pas inférieur à "Châtelet" dans l'ordre lexicographique) :

```
ALTER TABLE distance ADD CONSTRAINT asym CHECK(destS < sourceS)
```

Attention par la suite à bien utiliser des disjonctions dans les requêtes pour tenir compte de cette asymétrie.

Il est également possible d'utiliser la vue suivante `distance_vue` à la place de la table `distance` dans les requêtes :

```
CREATE VIEW distance_vue AS
           (SELECT * FROM distance
                      UNION
           SELECT sourceS as destS, destS as sourceS, meters FROM distance);
```

L'utilisation de cette vue ne dispense évidemment pas de créer en premier lieu la contrainte de vérification obligeant la table `distance` à être asymétrique. La vue ne fait que restaurer virtuellement la symétrie. Contraindre en revanche la table `distance` à être symétrique doublerait inutilement la taille de celle-ci et requerrait un trigger afin d'assurer la vérification de la contrainte de symétrie lors d'éventuelles insertions et mises à jour de la table.

Dans les trois prochaines questions, **on suppose que nos données satisfont la contrainte d'intégrité suivante** : pour toute station `idS` desservie par une ligne `line`, il existe toujours au moins un tuple de `route` dans lequel `destS` a pour valeur `idS`, ainsi qu'un autre tuple dans lequel `sourceS` a pour valeur `idS`. Notez que ceci vous permettra incidemment de simplifier l'écriture de vos requêtes.

Exercice 3 : requêtes SQL

Proposez des requêtes SQL permettant d'extraire les informations demandées.

1) La liste des noms de stations pour lesquelles il n'y a pas d'information quant à l'accessibilité aux personnes à mobilité réduite.

Solution:

```
SELECT nameS FROM station WHERE prm IS NULL;
```

2) La liste des lignes dont au moins une station est accessible aux personnes à mobilité réduite, sans doublon.

Solution:

```
SELECT distinct line FROM route WHERE destS IN (SELECT idS FROM station where prm)
```

Ou :

```
SELECT distinct line FROM route WHERE sourceS IN (SELECT idS FROM station where prm) ;
```

3) Les deux stations les plus éloignées l'une de l'autre.

Solution:

```
SELECT destS, sourceS FROM distance WHERE meters = (SELECT MAX(meters) FROM distance);
```

ou

```
SELECT destS, sourceS FROM distance WHERE meters >= ALL (SELECT meters FROM distance);
```

4) La liste des lignes qui ne desservent aucune station accessible aux personnes à mobilité réduite.

Solution:

```
SELECT line FROM line L
WHERE NOT EXISTS
(SELECT * FROM route R, station S
WHERE R.line=L.line
AND R.sourceS=S.idS
AND S.prm);
```

5) Pour chaque station, le nombre de lignes par lesquelles elle est desservie (tableau résultat : (`station`, `nb`)).

Solution:

```
SELECT sourceS as station, COUNT(distinct line) as nb
FROM route
GROUP BY  sourceS ;
```

6) Pour chaque ligne comptant au moins 10 stations, la station la plus centrale, i.e., desservie par le plus de lignes (tableau résultat : (`line`, `station`)). Les résultats doivent être triés dans l'ordre décroissant du nombre total de lignes desservies.

Solution:

```
WITH nblines AS (SELECT line, sourceS as station, COUNT(distinct line) as nb
                 FROM route
                 GROUP BY  line, sourceS
                 HAVING COUNT(distinct line)>10)
SELECT line, station FROM nblines N1
WHERE nb >= ALL (SELECT N2.nb FROM nblines N2 WHERE N1.line=N2.line)
ORDER BY nb DESC;
```

Exercice 4 : vues et requêtes récursives avec postgres

Un mouvement social intense s'empare de la France. En cette période de grève générale, seules les lignes automatiques fonctionnent. Dans ce contexte, la table `route` ne peut plus être utilisée de manière fiable pour le calcul d'itinéraire. À la place, un stagiaire créé et utilise la vue suivante :

```
CREATE VIEW social_traitre AS
(SELECT idR, line, sourceS, destS, deptime, arrtime
FROM line, route
WHERE idL=line
AND auto);
```

1) Proposez une requête SQL retournant les stations accessibles pendant la grève depuis la station Châtelet (dont l'identifiant de station est 66) via un nombre arbitraire de stations. Vous retournerez seulement les identifiants de station.

Solution:

```
WITH RECURSIVE Access(station) AS
(
SELECT destS FROM social_traitre WHERE sourceS=66
UNION
SELECT S.destS
FROM social_traitre S, Access A
WHERE S.sourceS = A.destS
)
SELECT *  FROM Access ;
```

2) La station Tuileries (dont l'identifiant de station est 42) prend feu suite à une émeute. Elle est donc fermée jusqu'à nouvel ordre. Notre stagiaire essaie alors de modifier la vue `social_traitre` afin de tenir compte de ce désagrément. Il utilise la requête suivante :

```
DELETE FROM social_traitre WHERE sourceS=42 OR destS=42;
```

Que se passe-t-il ? Expliquez-lui comment améliorer sa solution.

Solution: La mise à jour de la table `route` via la vue est refusée puisque la vue comporte une jointure. On préconise au stagiaire de mettre directement à `jour` la table route (s'il dispose des droits nécessaires, évidemment), ou mieux, de créer une nouvelle vue. On note que dans les deux cas, il y aura de toutes façons des soucis pour le calcul d'itinéraire, car en pratique la connection entre Concorde et Palais Royal musée du Louvre ne sera vraisemblablement pas perdue, le train ne marquera juste pas d'arrêt à Tuileries. Les modifications requises sur la table `route` seront donc plus complexes que de simples suppressions de tuples.

Exercice 5 : requêtes en algèbre relationnelle

Proposez des requêtes d'algèbre relationnelle permettant d'extraire les informations demandées.

1) Lee nom des stations dans lesquelles il y a des trains dès 05 :00 du matin.

Solution:

$$\pi_{nameS}(station \bowtie_{idS=sourceS} \sigma_{deptime<=05:00}(route))$$

2) Les lignes dont aucune station n'est accessible aux personnes à mobilité réduite.

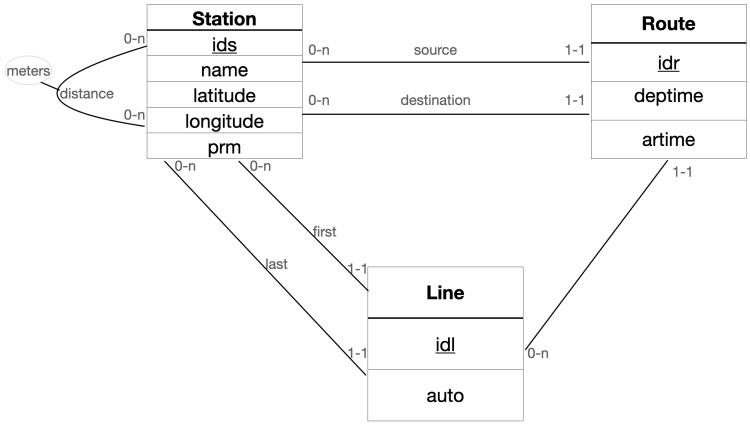
Solution:

$$\rho_{idL\to line}(\pi_{idL}(line)) - \pi_{line}(\sigma_{prm}(station) \bowtie_{idS=sourceS} route)$$

2 Modélisation : reverse engineering, enrichissement et amélioration du schéma

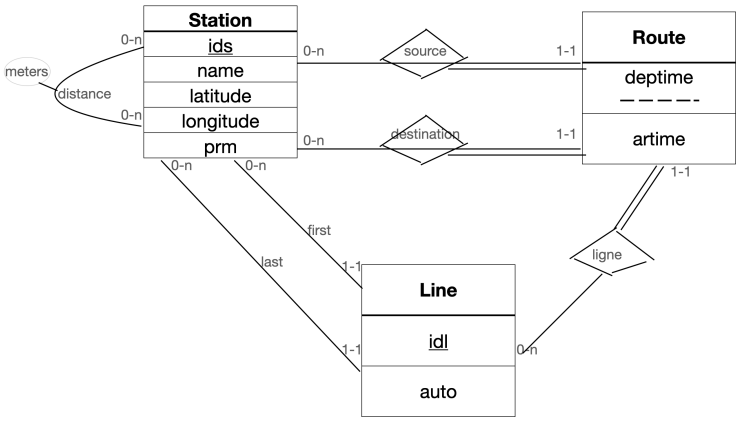
1. Proposer une modélisation E/R correspondant au schéma relationnel décrit précédemment. On précisera les entités, les identifiants, les associations, on indiquera pour chaque association les cardinalités. Listez les contraintes externes avec soin et expliquez comment les implémenter lorsque ceci est possible avec les notions vues en cours. Vous pouvez si nécessaire critiquer ce schéma et suggérer comment l'améliorer (par exemple, le choix de clef pour `route` est-il vraiment judicieux ?).

Solution: (On fera ici abstraction des grandes qualités esthétiques des diagrammes.)
Le diagramme suivant constitue un reverse engineering possible du schéma de l'énoncé. Les attributs `deptime` et `artime` pourraient être disposés au niveau des associations source et destination (un nom d'attribut `time` moins spécifique pourrait alors être utilisé). Les deux modélisations sont équivalentes.



La question était ouverte et de nombreuses améliorations du diagramme étaient possibles. Par exemple, rien n'indique la directionalité des routes et les attributs `firstS` et `lastS` n'ont pas un statut très satisfaisant... La correction de ce souci est laissée en exercice au lecteur. Pour saisir toute la complexité inhérente à ce type de cas pratique, vous pouvez analyser le schéma du TP8 (sur le réseau des transports d'Ile de France).

On propose ici simplement de corriger le problème flagrant posé par l'identifiant de route. Celui-ci est un potentiel facteur d'incohérence, puisqu'il permet deux passages d'un métro à la même heure dans une même gare avec des arrivées à des heures différentes à la même gare. La notion d'entité faible est définitivement requise ici (la situation est similaire à celle de l'entité voyage du projet). On propose :



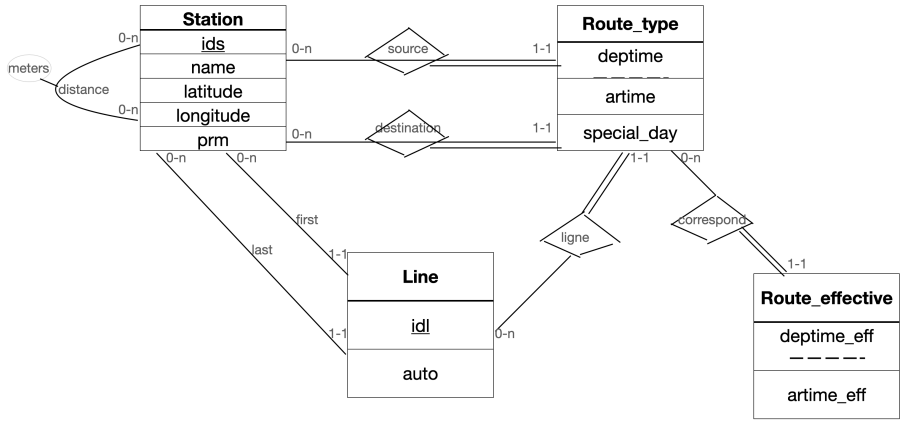
Notez que `(source, destination, line, deptime)` sera la clef primaire de ce qui deviendra la table `Route`, ce qui corrige notre problème initial d'incohérences potentielles. Remarquez également que contrairement à ce qui a été indiqué sur certaines copies, prendre comme clef primaire un sur-ensemble de `idr` tel que `(idr, source, destination)` ne corrige en rien le problème. Il n'est pas judicieux non plus de considérer `(source, destination)` (ceci induirait un seul passage par jour au plus entre les deux stations...). L'attribut `deptime` doit être ajouté comme

discriminant. Attention, considérer `(deptime, artime)` comme discriminant nous exposerait à nouveau à des incohérences. Morale : dans le contexte des bases de données relationnelle, il est fondamental de choisir soigneusement ses clefs.

Il y a tout un ensemble de contraintes externes, e.g., : `deptime < artime`, la distance entre A et B doit être la même qu'entre B et A. Il ne devrait pas y avoir de distance enregistrée entre A et A. Les tupls de route ne concernent que des stations adjacentes. Les données de latitudes et longitude doivent être cohérentes avec les données de distance enregistrées (mais de manière seulement approximative évidemment, puisque les chemins empruntées par les métros entre les stations sont rarement parfaitement droits), etc.

2. Les métros ayant récemment été équipés de traqueurs GPS, il a été décidé en hauts lieux d'enregistrer leurs horaires de passage effectifs afin de surveiller leur ponctualité. Un aménagement plus rationnel des horaires a par ailleurs été décidé : les trains rouleront plus tard tous les vendredi, samedi et veilles de jour férié. Modifiez le schéma E/R que vous avez proposé afin d'accommoder ces nouveaux besoins. N'oubliez pas de lister les contraintes externes.

Solution: On note la ressemblance avec le cas pratique considéré en tp dans lequel on distingue types de vol et vols effectués. Le cas de la bibliothèque est assez similaire (livre versus exemplaire). On peut proposer :



L'attribut `special_day` sera typé comme Booléen. Attention, pour `deptime, artime` les types utilisés seront différents de ceux de `deptime_eff, artime_eff` . Dans le second cas on utilisera de préférence des timestamp.

3. Proposez une traduction de votre diagramme dans le modèle relationnel. Précisez les clefs primaires et les contraintes référentielles (en particulier les clefs étrangères).

Solution: Schéma logique :

```
station(idS, nameS, lattitude, longitude, prm)
line(idL, firstS, lastS, auto)
route_type(line,sourceS,destS,deptime, artime, special_day)
route_effective(line,sourceS,destS,deptime,deptime_eff,
arrtime_eff)
distance(destS, sourceS, meters)
```

Clefs étrangères : `firstS`, `lastS` dans `line`, `destS`, `sourceS` dans `route` et dans `dis-`

`tance` font référence à `idS` dans `station` ; `line` dans `route` fait référence à `idL` dans `line`. `line,sourceS,destS,deptime,deptime_eff` dans `route_effective` fait référence à la clef primaire de `route_type`.

Parmi les autres contraintes, la plupart des attributs non clefs peuvent être déclarés non nulls, sauf peut être `latitude, longitude, prm, auto, meters` (en fonction des réquisits du client). On peut imposer également au niveau de la table distance `CHECK(destS < sourceS)`. La détermination d'un choix judicieux pour les types est laissé en exercice au lecteur.

PostScriptum : une solution alternative (et peut être même préférable en termes de gestion des index !) aurait consisté à garder la clef de route choisie initialement mais à déclarer le tuple d'attributs `(line,sourceS,destS,deptime)` de `Route` comme unique, c'est tout à fait valide aussi ! L'important pour l'examen était de noter le risque d'incohérences potentielles...