Partiel - Mars 2023

1 Partie 1 (dans une feuille à part)

1.1 A propos d'une station de ski - Modèle relationnel en compréhension

On étudie une application de base de données utilisée par une station de ski. Une station de ski souhaite construire une base de données pour enregistrer les informations sur son domaine skiable. Chaque piste a un nom, une difficulté (vert/bleu/rouge/noir), et une longueur. Une remontée a un nom et peut mener à plusieurs pistes. Une piste peut être desservie par plusieurs remontées (c'est à dire, que la remontée permettra accéder à certaines pistes). Les remontées sont organisées par altitude (haute/moyenne/basse), et toutes les remontées d'une même altitude ont le même horaire d'ouverture et de fermeture. Ces horaires sont mis à jour quotidiennement.

Pistes (nomP, difficulte, longueur)
/ <n, d, l> ∈ *Pistes* ⟺ *la piste de nom n a une difficulté d et une longueur l. */*
Remontees (nomR, nomP, altitude)
/ <n, p, a> ∈ *Remontees* ⟺ *la remontée de nom n et d'altitude a dessert la piste p */*
Horaires (altitude, ouverture, fermeture)
/ <a, o, f> ∈ *Horaires* ⟺ *les remontées d'altitude a ouvrent à l'heure o et ferment à l'heure f */*
Les domaines :
domaine(nomP)=domaine(nomR)= chaîne de caractères
domaine(difficulte) = { vert, bleu, rouge, noir}
domaine(altitude) = {haute, moyenne, basse}
domaine(longueur) = entier
domaine(ouverture) = domaine(fermeture) = date (à la minute près)

Question 1 (2 points) :
Indiquer les contraintes d'intégrité référentielle pour chaque relation. Expliquer avec une phrase en français ce qu'elles veulent dire.

Les contraintes d'intégrité :

Une piste desservie par une remontée doit être une piste existante (apparaît dans la table Pistes)
Remontees[nomP] ⊆ Pistes[nomP]
Une altitude mentionnée dans une remontée doit être altitude déclarée dans la liste d'Horaires
Remontees[altitude] ⊆ Horaires[altitude]

Question 2 (2 points) :
Donner le code SQL permettant de créer la table Remontees

```sql
-- On prend ici une syntaxe Oracle, PostGres ou MySQL
-- (en utilisant varchar et considérant les PKs comme UNIQUE + NOT NULL)
CREATE TABLE Remontees (
   nomR VARCHAR (50),
   nomP VARCHAR (50),
   altitude VARCHAR(6),
   CONSTRAINT pk_remontees PRIMARY KEY (nomR, nomP),
   CONSTRAINT fk_remontees_nomp FOREIGN KEY (nomP) REFERENCES Pistes(nomP),
   CONSTRAINT fk_remontees_altitude FOREIGN KEY (altitude) REFERENCES Horaires(altitude)
);
```

Question 3 (2 points) :
Imaginer que nous voulons sauvegarder l'information sur le fait qu'une piste communique avec une autre piste. Par exemple, les pistes Glacier 2 et Glacier 3 communiquent avec la piste Glacier 1. Proposer une relation avec sa spécification et ses contraintes permettant d'exprimer cela.

Risque de symétrie ici. On peut aussi garder de la forme "source" et "cible".
Communications (nomP1, nomP2)/* *clé composé car association many-to-many* */
/* <p1, p2> ∈ *Communications* ⟺ *la piste p1 communique avec la piste p2. Pour éviter la symétrie et le fait de pouvoir faire communiquer une piste avec elle-même on ajoute la condition nomP1 < nomP2* */
Communications[nomP1] ⊆ Pistes[nomP]
Communications[nomP2] ⊆ Pistes[nomP]

Question 4 (4 points) :
En considérant les relations fournies en annexe, donner **le résultat retourné** par chacune des requêtes ci-dessous (considérer le système Oracle utilisé en TP via Caséine) ainsi qu'**une phrase qui décrit le but de la requête** (si elle est correcte). Si la requête contient une erreur qui empêche l'exécution, l'indiquer. Dans tous les cas, sauf à cause d'une erreur d'exécution, fournir les n-uplets affichés sous forme de tableau.
Exemple d'erreur d'exécution :

```
-- Erreur: mauvais utilisation de la jointure. On aurait du utiliser USING à la place de ON
-- En plus il aurait fallu mettre un DISTINCT car potentiellement on peut avoir des doublons
SELECT nomP
FROM Pistes JOIN Remontees ON (nomP);
```

1.
```
SELECT MAX(longueur) AS maxL
FROM Pistes JOIN Remontees USING (nomP)
WHERE altitude = 'haute' AND
      difficulte='bleu';
```
Donne la longueur maximale des pistes bleues en haute altitude
```
maxL
-----
1200
```

2.
```
SELECT nomP
FROM Pistes
WHERE nomP NOT IN (SELECT *
                   FROM Remontees
                   WHERE altitude = 'basse');
```
La requête est incorrecte car les schémas dans le IN sont incompatibles.

3.
```
SELECT difficulte, SUM(longueur)
FROM Pistes
GROUP BY difficulte;
```
Donne la longueur totale des pistes pour chaque difficulté de piste
```
difficulte SUM(longueur)
-------------------------
bleu       7400
noir       666
rouge      2850
vert       750
```

4.
```
SELECT DISTINCT nomR
FROM Remontees JOIN Pistes USING (nomP)
WHERE difficulte = 'rouge';
```
Donne les remontées qui desservent au moins une piste rouge.
```
nomR
-----------------
Jandri Express 2
Pierre Grosse
```

2 Partie 2 (dans une feuille à part)

2.1 A propos d'une station de ski - Expression de requêtes en SQL

Question 5 (10 points) :
Exprimer en SQL les requêtes ci-dessous en respectant rigoureusement les contraintes données. Les

requêtes devront construire des résultats sans répétition de valeurs, la clause DISTINCT ne sera utilisée que lorsque nécessaire. Les produits de relation seront exprimés dans la clause FROM des requêtes.

1. Donner les noms de remontées qui desservent la piste 'Toura 1' (ex., Jandri Express 1)

```
SELECT nomR
FROM Remontees
WHERE nomP = 'Toura 1';
```

2. Pour chaque piste bleue avec au moins 2 remontées qui la desservent, donner son nom, sa longueur et le nombre de remontées qui la desservent (ex., Toura 1, 800, 3)

```
SELECT nomP, longueur, COUNT(nomR)
FROM Remontees JOIN Pistes USING (nomP)
WHERE difficulte = 'bleu'
GROUP BY nomP, longueur
HAVING COUNT(nomR) >=2;
```

3. Donner les pistes avec leur difficulté qui ne sont accessibles par aucune remontée (ex., Glacier 2, bleu).

```
SELECT nomP, difficulte
FROM Pistes
WHERE nomP IN (
   SELECT nomP
   FROM Pistes
   MINUS
   SELECT nomP
   FROM Remontees);

   ou

SELECT nomP, difficulte
FROM Pistes
MINUS
SELECT nomP, difficulte
FROM Pistes JOIN Remontees USING (nomP);
```

4. Donner les couples de pistes qui ont au moins 1 remontée en commun (ex., Diable 1, Jandri 1)

```
SELECT DISTINCT R1.nomP AS piste1, R2.nomP AS piste2
FROM Remontees R1 JOIN Remontees R2 ON (R1.nomP < R2.nomP AND R1.nomR=R2.nomR);
```

5. Donner les noms des remontées avec les pistes concernées, leur difficulté et leur longueur, pour les remontées fermant le plus tard (ex. Diable, Petites Crêtes, vert, 750).

```
SELECT nomR, nomP, difficulte, longueur
FROM Remontees JOIN Horaires USING (altitude) JOIN Pistes USING (nomP)
WHERE fermeture IN (
   SELECT MAX(fermeture)
   FROM Horaires);
```