Cours

Programme:

- Tables, relations, attributs / collonnes, domaine, schéma de table, enregistrements ou lignes, types de données
- Clé primaire, secondaire, étrangère
- Entités et association, association $1 - 1$, $1 - *$, $* - *$. Séparation de $* - *$ en deux $1 - *$
- `SQL` : SELECT, WHERE, AS, projections, DISTINCT, LIMIT, OFFSET, ORDER BY, UNION, INTERSECT, EXCEPT, produit cartésien, opérations au programme
- Jointures internes JOIN, JOIN ON, LEFT JOIN ON
- Agrégats: MIN, MAX, SUM, AVG, COUNT
- HAVING

Map d'une requête:

```
SELECT (DISTINCT) champs -- `champs` peut avoir des agrégas
AS names
FROM data
(LEFT|INNER) JOIN jointures ON eq
WHERE condition    -- filtre les données avant regroupement
GROUP BY attributs -- fabrique les groupes
HAVING condition   -- filtre les groupes
ORDER BY attributs -- tri par `attributs` dans l'ordre lexico
LIMIT nombre       -- prendre seulement `nombre` lignes
OFFSET nombre      -- ignorer les premières `nombre` lignes
```

Films et notes[1]

On considère une bases de donn´ees contenant trois tables:

- `Realisateurs(id, nom, prenom, nation, nb films)`
- `Films(id, nom, id_realisateur)`
- `Notes(id, id_film, note)`

1. Pour chaque table, donner les clés primaire et étrangère
2. Écrire une requête SQL qui retourne les noms et prénoms des réalisateurs américains ayant fait plus de 10 films
3. Écrire une requête SQL qui retourne le réalisateur ayant fait le plus de films
4. Écrire une requête SQL qui retourne la table des filmss avec le nom et prénom du réalisateur
5. Écrire une requête qui retourne la note moyenne reçu par un film ayant été réalisé oar nu américain

[1] Mallory Marin

Des étudiants et des notes[2]

On considère une bases de donn´ees contenant 2 tables:

- `Etudiants (id_etudiant, prenom, nom)`
- `Examen(id_examen, id_etudiant, matiere, note)`

1. Donner pour chaque table les clés primaire et les clef secondaires
2. Donner un diagramme entité-association correspondant à cette base de données relationnelle
3. Écrire une requête SQL qui retourne l'ensemble des examens pour laquelle la note est plus grande que 15
4. Écrire une requête SQL qui retourne la table contenant pour chaque étudiant son identifiant, nom, prénom, et moyenne générale.
5. Écrire une requête SQL donnant les noms et prénoms des étudiants ayant au moins eu une note supérieure à 15.

[2] Mallory Marin

Matières difficiles

On considère une base de données contenant 3 tables:

- `Etudiant(id, nom)`
- `Matiere(id, nom, difficulte)`
- `Note(id_etudiant, id_matiere, note)`

Une matière est difficile si `difficulte >= 4`. Un étudiant a validé une matière si sa note est au moins 10.

1. Trouver les étudiants à qui il manque au moins une matière difficile.
2. Trouver les étudiants qui ont validé toutes les matières difficiles.
3. Trouver tous les étudiants qui ont validé toutes les matières difficiles et aucune matière facile (de note ≤ 2).

Ensembles Egaux

On considère une base de données contenant 2 tables:

- `Document(id, titre)`
- `Mot(document, mot)`

On identifie un document à l'ensemble des mots qu'il contient, sans tenir compte des répétitions.

1. Trouver tous les couples de documents distincts (d_1, d_2) tels que tous les mots de d_1 apparaissent dans d_2.
2. Trouver tous les couples de documents distincts (d_1, d_2) tels que d_1 et d_2 ont exactement le même ensemble de mots.
3. Trouver les classes de documents équivalents (i.e. qui ont l'exact même ensemble de mots), représentées par le plus petit identifiant de la classe.

La limite de SQL

On montre ici qu'il existe des requêtes décidables qui ne sont pas exprimables dans le fragment de SQL vu au programme.

On considère pour cela la base de données composé d'une seule table `Arete(x,y)` représenant un graphe orienté avec potentiellement des boucles. On ne considère que des requêtes dans le fragment utilisant seulement les mots-clefs `SELECT`, `DISTINCT`, `FROM`, `JOIN`, `AS`, `ON`, `WHERE` et des conditions dans le WHERE.

Soit $G = (S, A)$ un graphe orienté, un sommet est dit *minimal* s'il n'existe pas de $y \in S$ tel que $(y, x) \in A$, et il est dit maximal s'il n'existe pas $y \in S$ tel que $(x, y) \in A$. On considère ici que le graphe donné contient un sommet initial.

1. Écrire une requête SQL qui donne la table des sommets initiaux et finaux.
2. Écrire une requête SQL qui renvoie les couples (x, y) tel qu'il existe un chemin de longueur exactement 2 entre x et y
3. On considère la requête suivante. Que reconnais-t'elle ? Montrer que l'on peut écrire une requête équivalente sans utiliser `JOIN`.

```sql
SELECT DISTINCT a.x, c.y
FROM Arete AS a
JOIN Arete AS b ON a.y = b.x
JOIN Arete AS c ON a.x = c.x AND b.y = c.y;
```

4. Expliquer comment transformer une requête utilisant des `JOIN` en une requête utilisant seulement des `SELECT` et `WHERE`

Dans la suite, on se place donc dans ce fragment: les requêtes sont écrites avec plusieurs occurrences de tables dans le `FROM`, puis des conditions d'égalité dans le `WHERE`.

Pour une requête R, on appelle taille en arêtes de R le nombre d'occurrences de la table `Arete` dans la clause `FROM`. Par exemple, la requête suivante a taille en arêtes 2 :

```sql
SELECT DISTINCT a.x, b.y
FROM Arete AS a, Arete AS b
WHERE a.y = b.x;
```

5. Soit R une requête de taille en arêtes k. Soit T le résultat de la requête sur un graphe $G = (S, A)$.

   Montrer que pour chaque tuple $(x_1, ..., x_n) \in T$, il existe un sous-graphe $G' = (S, A')$ avec $A' \subseteq A$ et $|A'| \leq k$ tel que $(x_1, ..., x_n)$ appartienne encore au résultat de R évaluée sur G'.

On suppose maintenant, par contradiction, qu'il existe une requête R de ce fragment qui renvoie exactement les couples (s, t) tels que:

- s est un sommet initial,
- t est un sommet final,
- il existe un chemin orienté de s vers t.

On note k la taille en arêtes de R.

6. On considère P_n le chemin orienté de longueur n. Montrer que dans P_{k+1}, le sommet 0 est relié au sommet $k + 1$ par un chemin.
7. En appliquant la question 6 à la requête R sur le graphe P_{k+1}, obtenir une contradiction.
8. Montrer que même en ajoutant `ORDER BY`, `LIMIT` ou `OFFSET`, la requête R n'est pas exprimable.

Topologie d'une montagne[3]

On considère une base de donné pour représenter des données topographiques avec l'unique table `Altitude(x, alt)`. `x` est une clef primaire et l'on sait que toutes les valeurs de 1 à N sont remplies, avec une valeur d'altitude `alt` qui n'est jamais `NULL`.

Dans cet exercice, on interdit l'utilisation de GROUP BY, plusieurs fois SELECT dans la même requête, ORDER BY, LIMIT, OFFSET, HAVING.

1. Écrire une requête qui renvoie la somme des altitudes des enregistrement correspondant à une position d'altitude strictement positive.
2. Écrire une requête qui renvoie le plus grand dénivelé entre deux positions consécutives qui sont toutes deux au dessus du niveau de la mer.
3. On dit qu'une position `x` est un *record depuis la gauche* si aucune position strictement plus à gauche n'a d'altitude supérieure ou égale. Écrire une requête qui renvoies les records depuis la gauche.
4. Pour chaque position `x`, renvoyer la position $y \leq x$ où l'altitude est maximale parmi les positions $\{1, ..., x\}$. En cas d'égalité, prendre la position la plus à gauche.
5. $(*)$ Écrire une requête qui renvoie le nombre de plateaux, à savoir les zones de une ou plusieurs positions consécutives de même altitude, telles que cette altitude est strictement plus élevée que les altitudes des positions immédiatement avant la position la plus à gauche du plateau, et immédiatement après la position la plus à droite du plateau.

[3] Inspiré du concours Général d'informatique 2022