Bases de Données

TD Noté – Départements et régions

radu.ciucanu@insa-cvl.fr

Pour rappel, le rendu de ce sujet compte pour 50% de la note finale. Le rendu sur Celene contiendra 2 fichiers : `NOM_Prénom.sql` pour la Partie 1 et `NOM_Prénom.pdf` pour la Partie 2, en utilisant votre nom et prénom.

Similairement, si vous travaillez en binôme, vos fichiers doivent être nommés `NOM1-Prénom1_NOM2-Prénom2`. Un seul rendu par binôme est suffisant.

Partie 1 : SQL

Pour cette partie, il faut rendre un fichier `NOM_Prénom.sql` qui contient toutes les commandes et les commentaires SQL pour les différents exercices.

Utilisez des commentaires pour chaque numéro d'exercice et pour les réponses aux éventuelles questions.

Votre fichier ne doit pas avoir des erreurs de syntaxe SQL si on l'exécute avec `.read` dans `sqlite3`.

1. Téléchargez l'archive `dept-files.zip` et désarchivez. Pour l'instant, nous considérons les fichiers `regions.csv` et `departements.csv`, qui contiennent des données sur les départements et les régions de France [1]. Analysez ces fichiers `csv` pour comprendre comment les données y sont stockées.

   Ensuite, donnez les commandes SQL nécessaires pour créer deux tables `regions (rid, nom, chefLieu)` et `departements (code, nom, prefecture, rid)`, en respectant les contraintes de type des attributs, ainsi que les contraintes de clés primaires et étrangères [2].

   Pour finir, populez ces deux tables avec les données des fichiers `csv`.

2. Donnez une requête SQL qui retourne le code et le nom du département qui a la préfecture à Bourges.

3. Donnez une requête SQL qui retourne, pour chaque département : le code, le nom, la préfecture et le nom de sa région. Par exemple :

```
| 01   | Ain            | Bourg-en-Bresse | Auvergne-Rhône-Alpes |
| 02   | Aisne          | Laon            | Hauts-de-France      |
| 03   | Allier         | Moulins         | Auvergne-Rhône-Alpes |
...
```

4. Donnez une requête SQL qui retourne les noms de régions (en ordre alphabétique) avec le chef lieu, ainsi que pour chaque département de la région : le code, le nom et la préfecture. Par exemple :

```
| Auvergne-Rhône-Alpes | Lyon     | 01   | Ain            | Bourg-en-Bresse |
| Auvergne-Rhône-Alpes | Lyon     | 03   | Allier         | Moulins         |
| Auvergne-Rhône-Alpes | Lyon     | 07   | Ardèche        | Privas          |
...
```

5. Donnez une requête SQL qui retourne le code, le nom et la préfecture de tous les départements de la région `Centre-Val de Loire`.

6. Nous considérons maintenant le fichier `voisins.csv` de l'archive. Créez une table `voisins(rid1, rid2)` avec les bonnes contraintes de clés primaires et étrangères, et populez cette table avec les données du fichier `csv`. Enfin, donnez une requête SQL qui retourne le nombre total de tuples dans la table.

1. Données extraites de Wikipédia
https://fr.wikipedia.org/wiki/Liste_des_d%C3%A9partements_fran%C3%A7ais
https://fr.wikipedia.org/wiki/R%C3%A9gion_fran%C3%A7aise

2. N'oubliez pas d'inclure la commande `PRAGMA foreign_keys = ON;` tout au début de votre script SQL pour s'assurer que les contraintes de clés étrangères sont appliquées par sqlite. Comme vous l'avez peut-être déjà remarqué, ce logiciel a choisi de ne pas enforcer les clés étrangères dans son comportement par défaut cf. https://www.sqlite.org/pragma.html#pragma_foreign_keys

7. La relation de voisinage est naturellement symétrique. Qu'est-ce que vous remarquez concernant ce qui est stocké dans la table `voisins` ? Afin de simplifier la prise en compte de la symétrie, proposez une view `voisinsSym` qui construit la clôture symétrique de la relation `voisins`. Enfin, donnez une requête SQL qui retourne le nombre total de tuples dans votre view et comparez avec le nombre obtenu à la question précédente.
8. Créez une nouvelle view `voisinsSymNoms`, qui utilise la précédente et qui contient les relations de voisinage entre des noms de régions et pas des identifiants. En utilisant cette view, comptez le nombre de voisins par région, triés par ordre décroissant. Les régions sans voisins doivent aussi apparaître dans le résultat, avec compteur 0. Par exemple :

```
| Centre-Val de Loire        | 6 |
| Auvergne-Rhône-Alpes       | 5 |
| Ile-de-France              | 5 |
...
| Martinique                 | 0 |
| Mayotte                    | 0 |
```

9. Nous utilisons aussi dans la suite un autre jeu de données réelles concernant les Zones Urbaines Sensibles (ZUS). Tout d'abord, analysez les formats dénormalisés dans lesquels ces données ont été publiées (HTML et Excel) [3] et formulez quelques critiques. Ensuite, considérons le fichier `zus.csv` de l'archive, qui a été extrait à partir de ce qui a été publié. Créez une table `zus(departement, commune, quartier)`, avec les bonnes contraintes de clés primaires et étrangères, et populez cette table avec les données du fichier `csv`.
10. Donnez une requête SQL pour trouver les ZUS qui se trouvent dans plusieurs départements alors qu'il s'agit du même quartier et des mêmes communes. Dans le `csv`, cette information est encodée en utilisant un code de département entre parenthèses dans l'attribut commune.

    Ensuite, critiquez ce choix de stockage des données et expliquez comment vous auriez fait mieux.
11. Créez une view qui sélectionne les ZUS qui se trouvent dans les communes qui sont préfectures de leur département. Liste non-exhaustive de subtilités auxquelles il faut faire attention : les arrondissements des grandes villes (Paris/Marseille/Lyon), les situations quand le ZUS est à cheval sur plusieurs communes dont la préfecture, etc. Ensuite, donnez la requête SQL qui utilise cette view afin de retourner le nombre de ZUS dans des communes préfecture ou dans d'autres communes, ainsi que le total :

```
| 751      | Total   |
| 246      | Oui     |
| 505      | Non     |
```

12. Donnez une requête SQL qui compte, en ordre décroissant, le nombre de ZUS de chaque département, en incluant aussi sa région dans le résultat. Attention à faire aussi apparaître les départements qui n'ont pas de ZUS, avec compteur 0. Par exemple :

```
| Nord                   | Hauts-de-France           | 49 |
| Seine-Saint-Denis      | Ile-de-France             | 36 |
| Rhône                  | Auvergne-Rhône-Alpes      | 30 |
...
| Lozère                 | Occitanie                 | 0  |
| Tarn-et-Garonne        | Occitanie                 | 0  |
```

13. Donnez une requête SQL qui compte, en ordre décroissant, le nombre de ZUS de chaque région. Par exemple :

```
| Ile-de-France             | 157 |
| Hauts-de-France           | 94  |
| Grand Est                 | 88  |
...
| Guyane                    | 4   |
| Mayotte                   | 1   |
```

14. Donnez deux formulations différentes pour la requête qui retourne les noms des régions pour lesquelles tous les départements ont au moins un ZUS.

3. https://sig.ville.gouv.fr/atlas/ZUS

— Une version en utilisant uniquement des `select/from/where` et des requêtes imbriquées avec `exists/not exists`. Pour y arriver, c'est judicieux de spécifier d'abord la requête en TRC (cf. exercice 5 de la Partie 2), et ensuite de faire les transformations logiques afin d'obtenir la formulation SQL.

— Une version en utilisant n'importe quel feature SQL (par exemple `count`), pas uniquement les features mentionnés dans la version précédente.

Partie 2 : Langages relationnels

Pour cette partie, il faut rendre un fichier `NOM_Prénom.pdf` (produit avec Latex) qui contient les réponses aux différents exercices.

1. Exprimez la requête 2 de la Partie 1 en
   (a) Algèbre relationnelle
   (b) TRC
   (c) DRC
2. Exprimez la requête 3 de la Partie 1 en
   (a) Algèbre relationnelle
   (b) TRC
   (c) DRC
3. Exprimez la requête 5 de la Partie 1 en algèbre relationnelle, de deux façons différentes, mais bien évidemment équivalentes.

   Comme vous le savez déjà, le but de l'algèbre relationnelle est de passer d'une requête SQL (donc non-procédurale) à une séquence de procédures qui recherche les données sur le disque et construit le résultat de la requête.

   On peut voir chacune de vos deux formulations en algèbre relationnelle comme une séquence de procédures où on évalue d'abord l'opérateur le plus imbriqué, ensuite on utilise ce résultat pour évaluer un autre opérateur, etc., jusqu'à ce que l'on ait évalué tous les opérateurs. Parmi vos deux formulations en algèbre relationnelle, laquelle pensez-vous que c'est plus efficace à évaluer et pourquoi ?
4. Exprimez la requête 8 de la Partie 1 (sans le tri) en algèbre relationnelle. N'hésitez pas à vous en servir de l'opérateur d'affectation de l'algèbre relationnelle, par exemple pour les différentes views.
5. Exprimez la requête 14 de la Partie 1 en TRC et faites les transformations nécessaires afin d'obtenir une formulation SQL qui utilise uniquement des `select/from/where` et des requêtes imbriquées avec `exists/not exists`.