Introduction

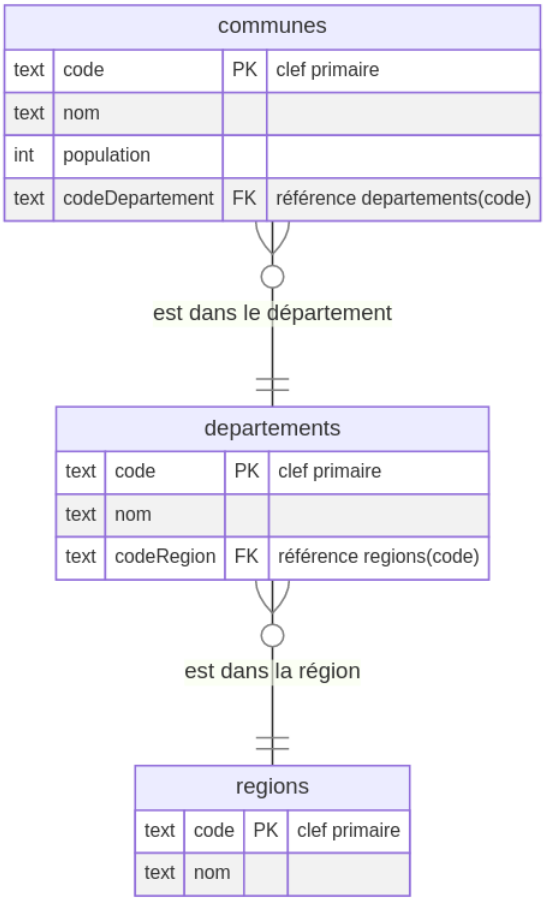
On commence par deux définitions importantes tirées de l'encyclopédie ouverte Wikipedia :

*En informatique, une interface de programmation d'application ou interface de programmation applicative (souvent désignée par le terme **API** pour **Application Programming Interface**) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle, un SGBD ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.*

Source : https://fr.wikipedia.org/wiki/Interface_de_programmation

*Les **données ouvertes** (en anglais : **open data**) sont des données numériques dont l'accès et l'usage sont laissés libres aux usagers, qui peuvent être d'origine privée mais surtout publique, produites notamment par une collectivité ou un établissement public. Elles sont diffusées de manière structurée selon une méthode et une licence ouverte garantissant leur libre accès et leur réutilisation par tous, sans restriction technique, juridique ou financière.*

Source : https://fr.wikipedia.org/wiki/Donn%C3%A9es_ouvertes



On trouve sur le Web de nombreux SGBD qui exposent des données ouvertes à travers une API Web. Par exemple, le site gouvernemental `https://geo.api.gouv.fr/` offre l'accès à une base ouverte de données géographiques nationales. Les requêtes sur la base se font avec le protocole HTTP avec des URL normalisées par l'API : par exemple `https://geo.api.gouv.fr/departements/69/communes` permet de récupérer une table des communes du département du Rhône au format JSON.
Dans ce TP, on commence par interroger cette base à travers son API pour récupérer des données sur les communes, régions et départements de France. Avec SQLite, on construit ensuite une base de données locale `communes-regions-departements.db` qui sera constituée de trois tables `communes`, `departements` et `regions`, dont on donne les schémas relationnels ci-dessus.

Consigne

- ☞ Extraire le fichier `materiel-tp1-sql.zip`.
- ☞ Modifier le script `geo_client_eleve.py` pour qu'il permette de créer la base `communes-regions-departements.db` après avoir interrogé l'API du site

  `https://geo.api.gouv.fr/`
- ☞ Charger la base `communes-regions-departements.db` dans `DB Browser`.
- ☞ Écrire chaque requête au brouillon, la tester en interrogeant la base avec `DB Browser`.
- ☞ Vérifier la requête dans le corrigé, appeler le professeur pour vérification en cas de doute.
- ☞ Lorsque la requête est validée, la recopier sur cet énoncé à l'emplacement laissé libre.
- ☞ Enregistrer la requête dans un fichier texte `tp1.sql` (onglet *Exécuter le SQL* dans `DB Browser`).

Exercice 1 *Exploitation d'une base de données ouvertes à travers une API*

1. Ouvrir puis exécuter le script `geo_client.py` dans un IDE Python.

Une erreur `OperationalError: no such table: communes` s'affiche, c'est normal.

Quel module est importé dans `geo_client.py` ? Afficher son interface avec la fonction `help`.

Contrainte importante : ne lisez pas l'implémentation du module importé, forcez-vous à le manipuler uniquement à travers son interface.

.....

.....

2. Quel est le type de l'objet `departements_data` créé ci-dessous ? et celui de son attribut `json_dico` ? Comment accéder à cet attribut ?

```
departements_data = geo.Georequest("https://geo.api.gouv.fr/
    departements/")
```

.....

.....

3. Compléter le corps de la fonction `creer_tables(base)` pour ajouter la table `communes` à la base de données `base` qui est un objet créé avec la classe `geo.Base`.

4. On suppose que la table `communes` a été correctement ajoutée dans la base. Exécuter la fonction `main()`. Comment interpréter l'erreur`IntegrityError: FOREIGN KEY constraint failed` d'après le cours du chapitre 1 *Modèle relationnel*?

.....

.....

5. Certaines communes récupérées par l'API avec :

   ```
   communes_data = geo.Georequest("https://geo.api.gouv.fr/communes/")
   ```

   ont un code de département `code_departement` et un code de région `code_region` qui ne correspondent pas aux données récupérées par l'API pour les régions et les départements. Il s'agit de communes des Territoires d'Outre-Mer qui n'appartiennent pas aux catégories administratives région et département, comme par exemple Nouméa :

   ```
   {'nom':'Nouméa','code':'98818','codeDepartement':'988','codeRegion':'
       988','codesPostaux':['98800'],'population':94285}
   ```

   Compléter le code de la fonction `client()` pour ne pas insérer dans la table `communes` ces valeurs qui rendent la base incohérente. Il suffit de filtrer celles qui ne sont pas dans la liste

   `liste_code_departements`.

   On va utiliser la base `communes-regions-departements.db` *ainsi créée.En cas d'échec, vous pouvez la récupérer dans le répertoire* `correction`.

Exercice 2 *Requêtes* `SELECT FROM WHERE`

1. Écrire une requête qui renvoie toutes les lignes de la table `communes`.

   .....

2. Écrire une requête qui renvoie toutes les lignes de la table `communes` ordonnées par population décroissante.

   .....

3. Écrire une requête qui renvoie uniquement les noms de toutes les lignes de la table `communes`.

   .....

4. Écrire une requête qui renvoie uniquement les noms de toutes les lignes de la table `communes` mais sans doublons.

.....

5. Écrire une requête qui affiche toutes les lignes de la table `communes` dont la valeur de la colonne `nom` est `'Belleville'`.

.....

6. Écrire une requête qui renvoie uniquement les colonnes `nom` et `codeDepartement` de toutes les lignes de la table `communes` dont la valeur de la colonne `nom` contient `'Belleville'` éventuellement suivie d'une autre partie.

.....

.....

.....

7. Écrire une requête qui affiche le nom et la population de tous les communes de la table communes dont le département est '2A' et la population est supérieure ou égale à 10000.

.....

.....

.....

8. Écrire une requête qui affiche dans l'ordre décroissant de la population, le nom et la population de tous les communes de la table communes dont le département est '2A' ou '2B' et la population est supérieure ou égale à 10000.

.....

.....

.....

Exercice 3 *Fonctions d'agrégations et alias de colonnes*

1. Écrire une requête avec la fonction d'agrégation `COUNT` qui renvoie le nombre total de communes françaises.

.....

2. Écrire une requête avec la fonction d'agrégation `COUNT` et le mot clef `DISTINCT` qui renvoie le nombre total de noms distincts de communes françaises

.....

3. Écrire une requête avec la fonction d'agrégation `SUM` qui renvoie la population totale du département `'69'`.

.....

4. Avec les fonctions d'agrégation `SUM`, `MIN`, `MAX`, `AVG` et le mot clef `AS`, écrire une requête qui, pour l'ensemble des communes françaises, renvoie une ligne avec quatre colonnes ;

- `pop_mini` contenant la population minimale ;
- `pop_maxi` contenant la population maximale ;
- `pop_total` contenant la somme de toutes les populations de communes ;
- `pop_moy` contenant la population moyenne par commune.

.....

.....

5. Pour déterminer toutes les communes dont la population est minimale par rapport à l'ensemble des communes françaises, on peut utiliser une *sous-requête* :

```
SELECT * FROM communes
WHERE population = (SELECT MIN(population) FROM communes);
```

À l'aide d'une *sous-requête*, écrire une requête qui renvoie toutes les communes du département `'92'` dont la population est minimale par rapport au sous-ensemble des communes du `'92'`.

.....

.....

.....

.....

Exercice 4 *Jointures simples*

1. Écrire une requête qui renvoie une nouvelle table constituée de trois colonnes :

- `nom_com` contenant le nom de la commune ;
- `nom_dep` contenant le nom de son département ;
- `population` contenant sa population.

Les lignes doivent être classées d'abord par ordre croissant de nom de département puis par ordre décroissant de population.

.....

.....

.....

.....

2. `LENGTH(chaine)` est une fonction `sqlite` permettant d'afficher la longueur d'une chaîne de caractères. Voir https://www.sqlitetutorial.net/sqlite-functions/sqlite-length/

   Écrire une requête qui renvoie une nouvelle table contenant les communes dont le nom a une longueur égale à la longueur du nom de leur département et constituée de deux colonnes :

   - `nom_com` contenant le nom de la commune ;
   - `nom_dep` contenant le nom de son département.

   Les lignes doivent être classées par ordre décroissant de nom de commune puis croissant de nom de département.

   .....

   .....

   .....

   .....

3. Précisez la requête précédente en sélectionnant uniquement les communes de plus de 100000 d'habitants et en ordonnant les résultats par population décroissante.

   .....

   .....

Exercice 5 *Jointures multiples*

1. Écrire une requête qui renvoie une nouvelle table constituée de quatre colonnes :

   - `nom_com` contenant le nom de la commune ;
   - `nom_reg` contenant le nom de sa région ;
   - `nom_dep` contenant le nom de son département ;
   - `population` contenant sa population.

   Les lignes doivent être classées d'abord par ordre croissant de nom de région, puis de département et enfin par ordre décroissant de population.

   .....

   .....

   .....

   .....

2. Écrire une requête qui renvoie le nombre de communes de la région `"Auvergne-Rhône-Alpes"`.

   .....

   .....

.....

.....

3. Écrire une requête qui renvoie la population totale de la région `"Auvergne-Rhône-Alpes"`.

.....

.....

.....

.....

4. Dans la table `regions`, la valeur de la clef primaire de la région `"Île-de-France"` est 11.

a. Écrire une requête qui renvoie le nombre de communes de plus de 50000 habitants dans cette région.

.....

.....

.....

.....

b. Écrire une requête qui renvoie le nom, le département et la population de la ou les commune(s) dont la population est minimale dans cette région.

.....

.....

.....

.....

5. Écrire une requête qui renvoie le nom, la population, le département et la région des communes dont la population totale est supérieure à celle de la région `"Mayotte"`. On utilisera une sous-requête.

.....

.....

.....

.....