1 Exploitation de données ouvertes

1.1 Qu'est-ce que des données ouvertes ou Open-Data

1.1.1 Définition

Les données ouvertes (Open Data en anglais) sont des **informations accessibles librement et gratuitement**, sous la forme de fichiers respectant des **formats interopérables**.

La finalité est de donner la possibilité à tout citoyen, toute entreprise ou association d'utiliser ces données numériques à ses propres fin d'analyse pour en extraire l'information désirée.

1.1.2 8 principes

Les données publiques sont considérées comme ouvertes si elles répondent à ces **8 principes** (2007, Open Government Data, USA)

1. **Complètes** : toutes les données doivent être rendues disponibles sauf les données pouvant porter atteinte à la vie privée des citoyens ou à la sécurité ;
2. **Primaires** : les données doivent être brutes, telles qu'elles ont été collectées à la source, non agrégées, non modifiées ;
3. **Récentes et actualisées** : elles doivent être rendues disponibles aussi vite que possible afin de préserver leur valeur ;
4. **Accessibles** : les données sont disponibles au plus large spectre d'utilisateurs ;
5. **Exploitables** : elles doivent être structurées et documentées afin de permettre un traitement informatisé ;
6. **Accès non discriminatoire** : elles sont disponibles à tout le monde de façon anonyme ne nécessitant pas d'enregistrement ;
7. **Format non propriétaire** : elles doivent être rendues disponibles au moins dans un format sur lequel aucune entité ne détient le monopole (ex : non PDF, non Excel) ;
8. **Libre de droits** : les données ne doivent pas être l'objet de droits d'auteurs, marques déposées, brevets, etc.

1.1.3 Origines des données ouvertes

Ces données ouvertes peuvent être :

- d'**origine publique** : émanant des services publics, de collectivités, de communes etc.
- d'**origine privée** : provenant d'entreprises et d'institutions dont les données concourent à des projets d'utilité publique, comme par exemple la SNCF, la RATP .. etc.

En France, la mission gouvernementale Etalab coordonne la mise à disposition de jeux de données ouvertes via le portail https://www.data.gouv.fr/fr/

Deux grands portails pour notre région :

- https://data.grandlyon.com/accueil
- http://opendata.auvergnerhonealpes.eu

1.2 Exploiter un jeu de données

Nous allons utiliser le jeu de données sur les **accidents de vélo** entre 2005 et 2018 :
https://www.data.gouv.fr/fr/datasets/accidents-de-velo/

1.2.1 Utilisation du tableur :

- Lire le descriptif de ce jeu de données
- Télécharger le fichier .csv
- L'ouvrir avec un tableur (éventuellement traiter les données pour les séparer en colonnes)
- Analyser les différents en-têtes de colonnes
- Donner le nombre d'accidentés

Ça se complique si l'on veut savoir par exemple le nombre de blessés légers en Isère en 2008 dans un virage…

Nous allons améliorer l'exploitation de toutes ces données en utilisant un SGBD sqlite3 :

1.2.2 Utilisation de DBBrowser pour créer une table sqlite3

Importer ce jeu de données en Open-Data dans une base de données Sqlite3 que vous nommerez *BDD_Accidents_Velos.sqlite3*. La table ainsi créée sera nommée *accidents*.

Pour cela, vous pouvez utiliser le tutoriel vidéo joint : **utilisation DB Browser.mp4**

1.2.3 Utilisation de DBBrowser pour exploiter des données

Question 1. Après importation des données, quel est le type des données pour l'attribut :
- *identifiantaccident*
- *typederoute*
- *departement*

Pour les questions suivantes, il faudra réaliser quelques requêtes simples que vous stockerez dans différents onglets du logiciel (que vous prendrez le temps de renommer avec le numéro de la question).

Question 2. Retrouver le nombre d'accidentés total sur la période.

Rajouter un alias à cette requête SQL pour que le tableau affiche *Nombre_Accidentés* en en-tête de colonne. (= renommage avec AS)

Question 3. Retrouver le nombre d'accidents total sur la période.

Rajouter un alias à cette requête SQL pour que le tableau affiche *Nombre_Accidents* en en-tête de colonne. (Ne pas compter les doublons)

Question 4. Donner le nombre d'accidentés dans le département du Rhône sur la période.

Question 5. Donner le nombre d'accidents où un cycliste a été tué sur la période, toujours dans le Rhône.

Question 6. Donner la date et le jour des différents accidents où un cycliste a été tué sur la période, toujours dans le Rhône.

Question 7. Reprendre la dernière requête pour afficher les données classées par ordre alphabétique des jours de semaine.

Question 8. Classer, par ordre **descendant**, l'ensemble des données par numéro du département. Quel est le problème repéré ? Est-ce cohérent avec le type de *departement* donné en question 1 ?

Question 9. On souhaite à présent savoir quand les accidents ont plutôt lieu dans la journée. On pourra utiliser une fonction d'agrégat. Proposer une requête SQL qui **permettrait d'avoir une idée**, n'oubliez pas de rajouter un ALIAS intelligible.

Question 10. Donner le nombre d'accidents par département, on classera ces résultats par nombre décroissant d'accidents. Pour cela, on utilisera la commande *GROUP BY* qui est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat

Voici une syntaxe générale de la commande :

```
SELECT colonne1, fonction(colonne2)
FROM table
GROUP BY colonne1
```

1.2.4 Exploitation des données visuellement

Question 11. On souhaite exploiter ces données visuellement sur une carte géographique avec des marqueurs pour placer les accidents. Quelles sont les 2 attributs qui vont devoir être utilisés ?

Question 12. Proposer une requête SQL qui affiche l'identifiant de l'accident, le département, la latitude et la longitude.

Question 13. En parcourant les données issues de la requête précédente, dire quels vont être les soucis rencontrés ?

1.3 Utiliser Python pour accéder à une base de données SQLite3

Le module *sqlite3* permet à un programme Python de se connecter à une base de données.

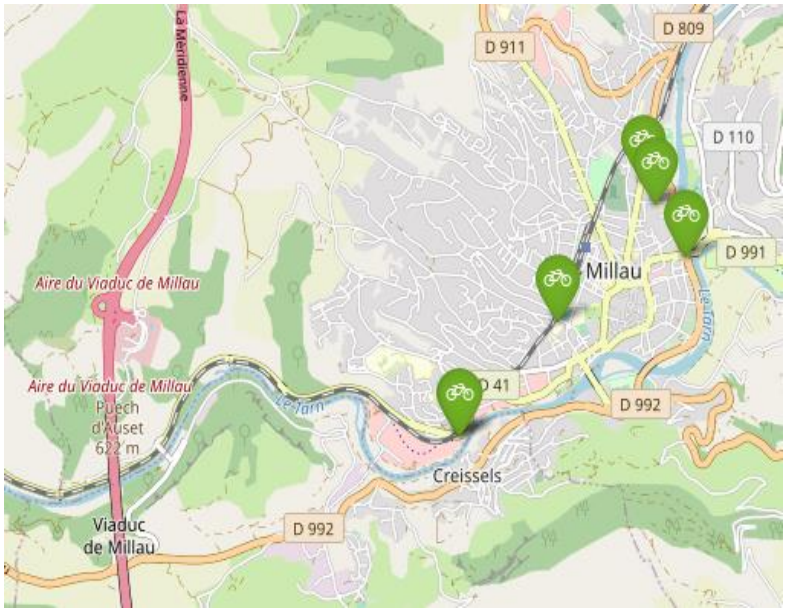
Voici un programme Python3 minimal pour ce besoin :

```
import sqlite3 # Import du module

# Connexion à la Base de données SQLlite
connexion_BD = sqlite3.connect(FICHIER_BDD)
c = connexion_BD.cursor()

#Exécution de la requête
c.execute(MA_REQUETE)

liste_data = c.fetchall()   #Récupération des résultats de la requête sous forme de liste
print("J'ai", len(liste_data), "enregistrements dans ma base de données")
```

Comprendre ce programme et le compléter afin qu'il interroge la base de données précédemment créée et que le terminal affiche la bonne valeur.

1.4 Créer une carte géographique avec des marqueurs

L'objectif est de créer une carte de ce type :



Chaque marqueur a nécessairement au moins 2 informations : une **latitude** et une **longitude**.

Nous allons utiliser le module Python *folium* pour ce travail. La documentation est disponible ici : https://python-visualization.github.io/folium/. Il s'installe via PIP avec la commande :

pip install folium

Pour fonctionner, il faut 3 choses :

1. Importer le module folium

2. Un objet *carte* que l'on créera avec l'instruction *folium.Map()*
3. Un ensemble de marqueurs que l'on créera avec l'instruction *folium.Marker()* qui faudra rajouter à la carte via la méthode *add_to()*.

Question 14. Création de la carte

En utilisant la documentation du module folium, créer une fonction *creation_carte()* définissant la carte.

Pour respecter la licence Open Database License (ODbL) des données `OpenStreetMap` qui s'affiche sur la carte, vous devez définir un argument à la carte `attr` (attribut fixé à `'© Contributeurs OpenStreetMap'`.

Les 2 autres arguments seront :

- `location` (pour centrer la carte sur un point, il s'agit d'un tuple, on pourra choisir *46.396, 2.505*)
- `zoom_start` (pour connaître le zoom de départ : entier compris entre 1 et 19)

Cet objet carte sera affiché dans un fichier html avec la méthode `.save(nom_fichier_carte.html)`

Vérifier le résultat dans le fichier HTML généré automatiquement.

Question 15. Ajout d'un marqueur

Au sein de la fonction précédente, rajouter un marqueur au centre de votre carte en complétant la méthode *folium.Marker()*.

Question 16. Personnalisation du marqueur

En lisant la documentation du module *folium*, personnaliser la couleur et le picto au centre du marqueur. On souhaite mettre une icône de vélo.

Question 17. Ajout de tous les marqueurs d'une liste

Soit la liste d'accidents suivante :

```
Liste = [
('accident1', 12, 44.29, 2.519),
('accident2', 12, 44.339, 2.105),
('accident3', 12, 43.855, 2.838),
('accident4', 12, 44.332, 2.787),
('accident5', 12, 44.126, 3.253),
('accident6', 12, 44.484, 2.496),
('accident7', 12, 44.316, 2.585),
('accident8', 12, 43.849, 2.899),
('accident9', 12, 44.366, 2.04199),
('accident10', 12, 43.933, 2.664)
]
```

Modifier la fonction *creation_carte()* pour faire en sorte que chaque accident soit un marqueur sur la carte. Vous devez passer en paramètres de la fonction, le nom de la liste et l'indice de la latitude et de la longitude dans cette liste.

Question 18. Créer une carte complète

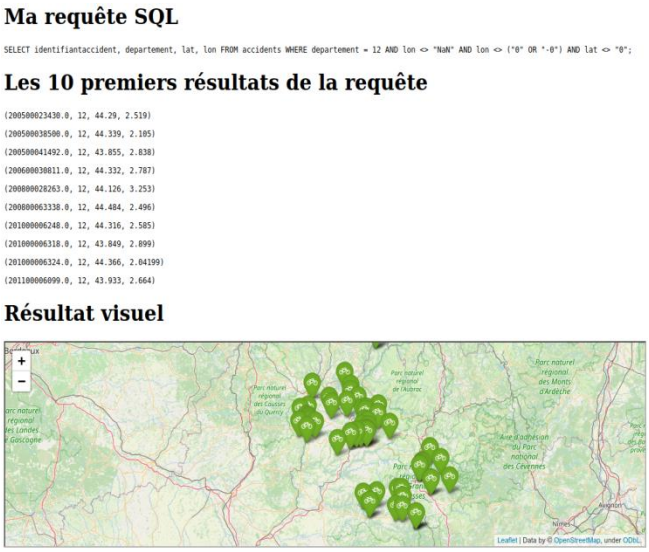
Créer une carte avec les 'vraies' données des accidents de vélos. Pour cela, on se limitera à un seul département. Rechercher dans un premier temps la requête avec DB Browser qui permet de ne récupérer que les accidents contenants des données de localisation.

Modifier ensuite le code pour créer la carte, pour le département 12 par exemple.

Question 19. BONUS : Page HTML de visualisation

Créer une page HTML de visualisation des requêtes SQL sur une carte comme le présente cette capture :

Le fichier HTML de cette page sera généré par Python. On pourra lui faire créer un fichier texte avec l'extension *.html* et afficher la carte OpenStreetMap dans une balise *iframe*.



1.5 Jointure de tables

Dans le fichier, nous ne disposons pas de l'information du nom de la commune, ce qui n'est pas pratique quand on souhaite connaître le nombre d'accidentés sur *Oullins* par exemple.

On va utiliser le jeu de données de la base officielle des **codes postaux** : https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/

Question 20. Import du jeu de données

En utilisant le tutoriel vidéo disponible ci-dessus ainsi que le déroulé de la partie précédente, importer, dans notre base de données, une nouvelle table *codes_postaux* à partir du fichier téléchargé.

Question 21. Identification

Dans les 2 tables *accidents* et *codes_postaux*, repérez les 2 attributs à rapprocher afin de pouvoir réaliser la jointure des 2 tables :

Question 22. Jointure

On souhaite connaître les accidents sur la période pour la commune d'*Oullins* (écrit en toutes lettres dans la requête). Proposez une requête SQL qui permet de renvoyer ces accidents.

Une piste…

On pourra réaliser une jointure entre les 2 tables sur les 2 attributs vus précédemment.

1.6 Normalisation de la base de données

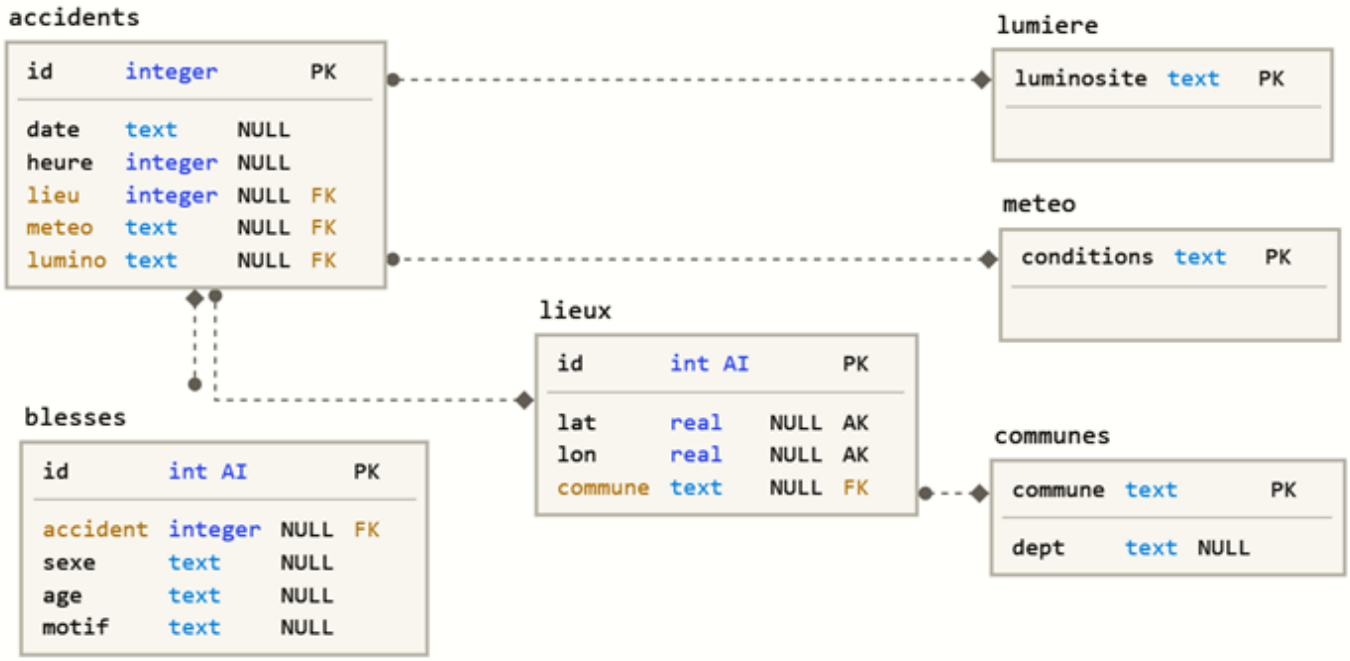
Ce fichier CSV téléchargé est très fourni en données mais certains attributs sont très peu utilisés, nous allons donc essayer de construire une base de données plus propre à maintenir sur le long terme.

En effet, dans les données originales, chaque enregistrement regroupe des données relevant soit de l'accident en lui-même (lieu, conditions…) soit du blessé (sexe, tranche d'âge, gravité des blessures…) mais sans vérifier qu'on n'associe pas par exemple un lieu différent à deux blessés du même accident.

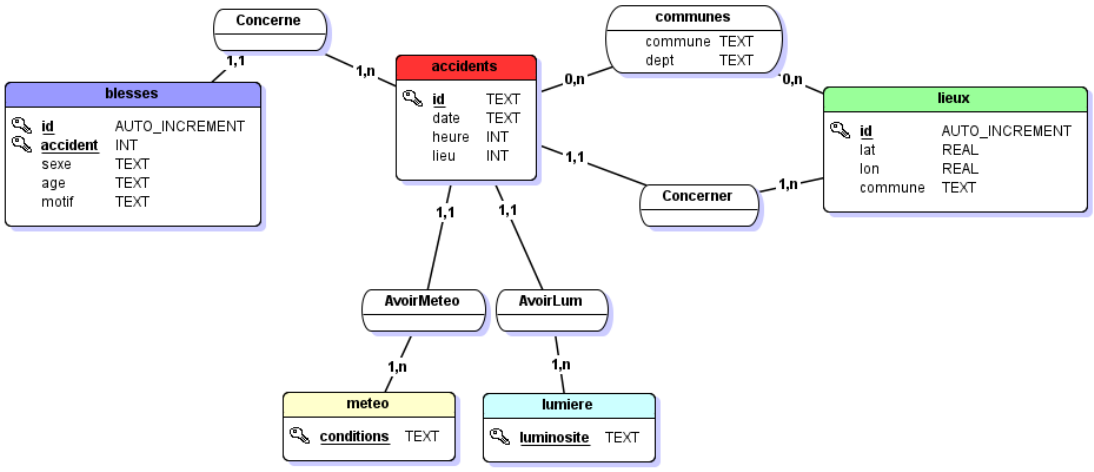
Pour éviter ce désagrément, nous organisons une partie des données originales en tables visant à séparer les données relevant des blessés et celles relevant de l'accident lui-même.

Dans cette partie, nous allons donc utiliser les dernières commandes SQL à connaitre pour cette année : *INSERT*, *UPDATE* et *DELETE*.

La base de données modélisée en plusieurs tables peut être représentée ainsi :



Autre représentation avec les cardinalités :



Exemple de lecture des cardinalités :

Un blessé concerne 1 et 1 seul accident.

Un accident concerne 1 ou n accident

1.6.1 Scripts de création de la base

Nous allons donc créer une nouvelle base de données puis on importera à nouveau les données du fichier CSV pour alimenter ces nouvelles tables.

Question 23. Créer une nouvelle base de données (vous pouvez fermer les autres) puis importer le fichier csv en créant une table nommée *accidents-velos*.

Question 24. Dans DBBrowser, exécutez le script SQL ci-dessous et constatez le travail réalisé dans la structure de la base de données.

```
DROP TABLE IF EXISTS lumiere;
CREATE TABLE lumiere (luminosite TEXT PRIMARY KEY);
```

Question 25. Ecrire les requêtes permettant de créer les tables meteo et lieux et les executer sur DBBrowser.

Question 26. Le script SQL ci-dessous crée l'ensemble des tables, executez-le et verifier le résultat sur DBBrowser.

```
DROP TABLE IF EXISTS lumiere;
DROP TABLE IF EXISTS meteo;
DROP TABLE IF EXISTS lieux;
DROP TABLE IF EXISTS communes;
DROP TABLE IF EXISTS accidents;
DROP TABLE IF EXISTS blesses;

CREATE TABLE lumiere (luminosite TEXT PRIMARY KEY);
CREATE TABLE meteo (conditions TEXT PRIMARY KEY );
CREATE TABLE communes (commune TEXT PRIMARY KEY, dept TEXT);
CREATE TABLE lieux (id INTEGER PRIMARY KEY AUTOINCREMENT, lat REAL, lon REAL, commune T
EXT REFERENCES communes(commune), UNIQUE (lat,lon));
CREATE TABLE accidents (id INTEGER PRIMARY KEY, date TEXT, heure INTEGER, lieu INTEGER
REFERENCES lieux(id), meteo TEXT REFERENCES meteo(conditions), lumino TEXT REFERENCES l
umiere(luminosite));
CREATE TABLE blesses (id INTEGER PRIMARY KEY AUTOINCREMENT, accident INTEGER REFERENCES
accidents(id), sexe TEXT, age TEXT, motif TEXT);
```

1.6.2 Suppression et Insertion des données

Les données Open-Data sont souvent partiellement fausses ou mal consolidées, c'est le cas ici.

1. des coordonnées GPS sont à 0 ou à NaN.
2. certaines coordonnées géographiques sont associées à plusieurs codes commune distincts.

Remarque : si vous n'êtes pas convaincu du point 2, vous pouvez exécuter cette requête et observer son résultat :

```
select T1.*, T2.commune FROM (select distinct lat, lon from (select lat, lon,
count(*) from (select distinct lat,lon,commune from 'accidents-velos') group by
lat,lon order by count(*)) as R WHERE R.'count(*)'=2) as T1 JOIN 'accidents-
velos' AS T2 using (lat,lon);
```

Pour régler le premier point, nous prenons le parti de supprimer ces données dans la table du fichier CSV, avant donc de les introduire dans les tables. C'est le sens de la première ligne du script ci-dessous.

Pour le second point, on procède au remplissage de la table lieux avec des requêtes imbriquées, qui permettent de sélectionner **seulement l'un** de ces codes commune par un choix arbitraire automatique.(on ne garde que la première commune associée à un couple de coordonnées, quand il y en a plusieurs)

Question 27. Dans DBBrowser, exécutez le script SQL ci-dessous(onglet *Exécuter le SQL*) et constatez le travail réalisé dans les différentes tables…

```
-- Suppression des données avec une latitude problématique
DELETE from 'accidents-velos' WHERE CAST(lat as REAL) < 0.10;

-- Remplissage des tables
INSERT INTO lumiere (luminosite) SELECT DISTINCT luminosite FROM 'accidents-velos';
INSERT INTO meteo (conditions) SELECT DISTINCT conditionsatmosperiques FROM 'accidents-
velos';
INSERT INTO communes (commune, dept) SELECT DISTINCT commune, departement FROM 'acciden
ts-velos';
INSERT INTO lieux(lat,lon,commune) SELECT lat,lon,min(commune) FROM (SELECT DISTINCT la
t,lon,commune FROM 'accidents-velos') GROUP BY lat,lon;
INSERT INTO accidents (id, date, heure, lumino, meteo, lieu) SELECT DISTINCT R.identifi
antaccident, R.date, R.heure, R.luminosite, R.conditionsatmosperiques, L.id FROM'accide
nts-velos' AS R JOIN lieux AS L USING (lat, lon);
INSERT INTO blesses (accident, sexe, age, motif) SELECT identifiantaccident, sexe, age,
motifdeplacement FROM 'accidents-velos';
```

Question 28. Expliquer cette requête la première requête du script qui nettoie la table source *accidents-velos*, `DELETE from 'accidents-velos' WHERE CAST(lat as REAL) < 0.10;`

Question 29. Un policier a mal saisi un accident dans le formulaire web. Nous souhaitons corriger l'horaire de l'accident *200500000623* et mettre qu'il a eu lieu à 10h.

Récupérer l'ancien horaire via une requête SQL et corriger-le via une nouvelle requête.

Question 30. Pour finir, nous allons utiliser la table *Lieux* que nous avons construite pour corriger les données initiales. Pour cela, on utilise la commande suivante :

```
UPDATE 'accidents-velos' SET commune = (SELECT lieux.commune FROM lieux WHERE lieux.lat
= 'accidents-velos'.lat and lieux.lon = 'accidents-velos'.lon);
```

Expliquer ce qu'elle fait.

Question 31. Saisir la requête SQL pour supprimer la table *accidents-velos*.

Question 32. Vous êtes allez trop vite ? Voilà de quoi prolonger votre plaisir…

Reprendre le travail permettant d'afficher la page HTML et l'adapter à la nouvelle structure de données.

Compléter celui-ci afin d'ajouter un formulaire (avec liste déroulante) permettant de choisir la commune pour laquelle afficher les accidents.

Vous pouvez également ajouter des filtres suplémentaires comme la météo, le degré de gravité de l'accident…