UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
iut
CRÉTEIL-VITRY
DÉPARTEMENT INFORMATIQUE

PREMIERE PARTIE – CREATION DE LA BASE DE DONNÉES

SAE 1.04

POUR LE 23/01/2024

Ahmed MEZGHICHE, Adel ACHOUCHE, Sami OURRAD, Dina NACHATE, Antonin JOMIE, Venancio ALVES

Tableau des matières

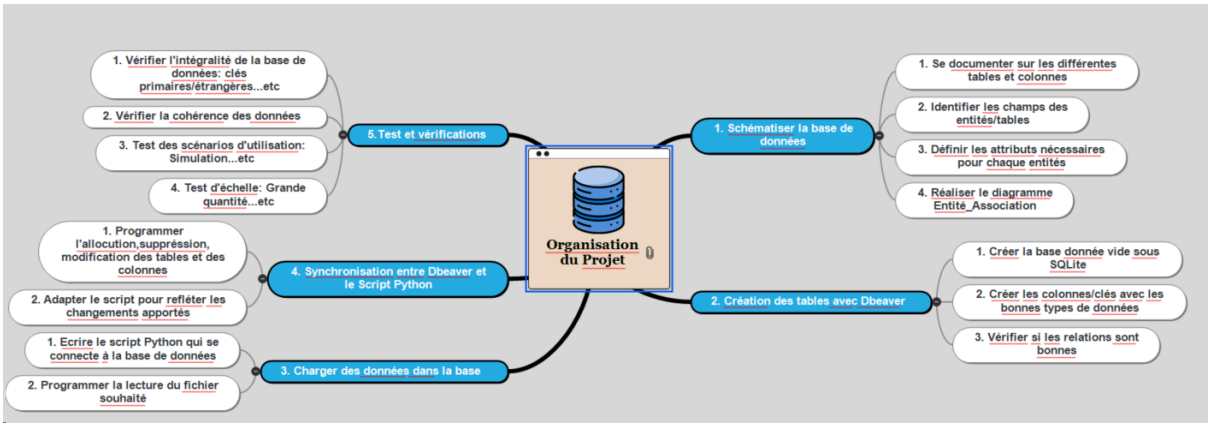
Introduction

La première étape lors du démarrage de notre projet consiste à concevoir et implémenter une base de données SQLite. L'objectif de cette initiative est de créer un environnement dans lequel les données mesurées à toutes les stations AASQA peuvent être entièrement stockées tout au long de la journée. L'objectif fondamental est de développer une base de données capable de gérer de manière précise et complète diverses informations concernant les mesures de polluants. Il est donc important de souligner que la création de cette base de données ne va pas sans problèmes et défis. Le principal défi est le fait de mettre en place une base de données de production. Ceci est essentiel pour la deuxième phase du projet, qui consiste à implémenter le code Python dans un framework web et à rendre la base de données disponible via le navigateur.

Notre projet a suivi un processus décomposé en cinq étapes clés. La première phase a commencé par l'illustration de la base de données et la définition de la structure de base du système. Cette approche a jeté les bases nécessaires pour d’autres processus. Après cette création de diagrammes, la deuxième étape consistait à créer une table à l'aide de DBeaver, un puissant outil de gestion de base de données. L'objectif était de transformer le modèle conceptuel en une infrastructure opérationnelle pouvant accueillir des données pertinentes pour le projet. Après avoir configuré la table, la troisième étape consiste à charger les données dans la base de données nouvellement créée. Cela a nécessité un soin particulier pour garantir l’intégrité des informations et garantir que la base de données était fonctionnelle. La quatrième étape a mis en évidence la nécessité de synchroniser DBeaver avec les scripts Python que nous avons développés. Cette synchronisation était essentielle pour garantir une interaction fluide entre la base de données et le code et garantir une exécution cohérente de l'application. La dernière étape était dédiée au test et à la validation du système. Cela comprend une

évaluation rigoureuse pour garantir que toutes les fonctionnalités fonctionnent et que les résultats obtenus répondent aux attentes initiales. Ainsi, ces cinq étapes successives ont constitué le cadre méthodique de notre projet, de la conception initiale à la réalisation concrète, garantissant une progression ordonnée et efficace à chaque étape du processus.

Pour concrétiser la première partie de notre projet, nous avons entrepris une démarche méthodique en créant une carte mentale complète. Cette représentation visuelle s'est avérée être un outil essentiel, nous offrant une vue d'ensemble structurée. À l'aide de cette carte, nous avons pu clairement définir l'ordre séquentiel des étapes à suivre, établissant ainsi une feuille de route claire pour l'avancement du projet.



Carte mentale pour l'organisation de notre projet.

Schématisation de la base de données

La conception de la base de données a nécessité une approche réfléchie et méthodique. Tout commence par une analyse minutieuse des fichiers CSV, chacun contenant un ensemble de données différent. La première étape est une inspection détaillée de chaque table contenue dans ces fichiers. Il s'agit d'un processus essentiel pour comprendre la richesse et la variété des informations disponibles.
Cette étape est également nécessaire pour respecter les conditions de création de la base de données et s'assurer de la faisabilité de l'idée. Une compréhension approfondie de la structure des données des fichiers CSV nous a permis de concevoir une base de données qui répond efficacement aux besoins et aux exigences de notre projet.
Notre schéma, élaboré à travers une démarche méthodique, se présente de la manière suivante:

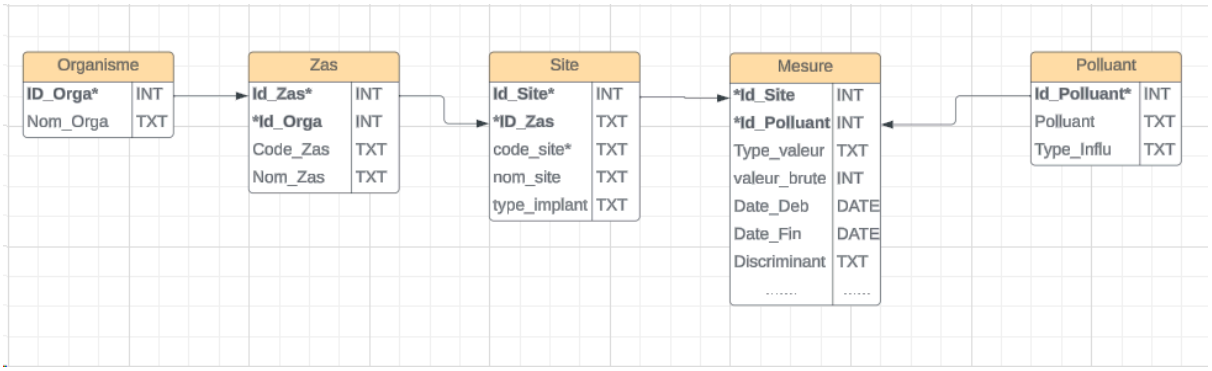


Schéma de la base de données.

Grâce à ce schéma détaillé, nous pouvons observer tous les composants du tableau et leurs attributs spécifiques. Cette représentation visuelle fournit un aperçu complet de la structure de notre base de données, facilitant la compréhension et la gestion précise de chaque entité et de ses caractéristiques associées.
Les tables que nous avons créées et leurs relations définissent la structure de notre base de données. Les clés, éléments cruciaux, sont utilisées pour établir des liens significatifs entre les différentes entités de notre base de données. Voici une explication détaillée de chaque table, mettant en avant l'utilisation des clés dans la structuration de notre base de données :

Création des tables avec DBeaver

L'élaboration de nos tables, élément central de notre base de données, a été rendue possible grâce à l'utilisation avisée du logiciel DBeaver. Cet outil de gestion de bases de données performant a été le moteur de notre démarche, nous permettant de concrétiser notre schéma conceptuel en une infrastructure opérationnelle. Dans les sections à suivre, nous plongerons dans une exploration détaillée de chaque table, mettant en lumière la contribution cruciale de DBeaver dans la création de ces structures interconnectées. Cette approche méthodique jette les bases nécessaires pour stocker de manière cohérente les données essentielles à notre projet

La table Organisme:

```
CREATE TABLE Organisme (
   Id_Orga INTEGER PRIMARY KEY,
   nom_orga TEXT UNIQUE
);
```

- **Id_Orga (INTEGER PRIMARY KEY) :** Identifiant unique de l'organisme, agissant comme une clé primaire. Il assure l'unicité de chaque organisme dans la table.
- **nom_orga (TEXT UNIQUE) :** Nom de l'organisme, avec une contrainte d'unicité, permettant une identification sans ambiguïté.

Cette table enregistre les informations sur les organismes, tels que les Agences de Surveillance de la Qualité de l'Air (AASQA). Chaque organisme a un identifiant unique (Id_Orga) et un nom (nom_orga).

La table Zas :

```
CREATE TABLE Zas (
      Id_Zas INTEGER PRIMARY KEY,
   Code_zas TEXT UNIQUE,
   Nom_zas TEXT UNIQUE,
   Id_Orga INTEGER,
   FOREIGN KEY(Id_Orga) REFERENCES Organisme(Id_Orga)
);
```

- **Id_Zas (INTEGER PRIMARY KEY) :** Identifiant unique de la ZAS, agissant comme clé primaire.
- **Code_zas (TEXT UNIQUE) :** Code identifiant la ZAS, avec une contrainte d'unicité, assurant ainsi une référence unique.
- Nom_zas (TEXT UNIQUE) : Nom de la ZAS.
- **Id_Orga (INTEGER) :** Clé étrangère liée à la table Organisme (FOREIGN KEY(Id_Orga) REFERENCES Organisme(Id_Orga)), établissant une relation entre les ZAS et les organismes.

Cette table enregistre les différentes Zones Administratives de Surveillance (ZAS), avec un identifiant unique (Id_Zas), un code spécifique (Code_zas), un nom distinctif (Nom_zas), et une référence à l'organisme correspondant.

La table Site :

```
CREATE TABLE Site (
      Id_Site INTEGER PRIMARY KEY,
   Code_site TEXT UNIQUE,
   Id_Zas integer,
   Nom_site TEXT UNIQUE,
   Type_impl TEXT ,
   FOREIGN KEY(Id_Zas) REFERENCES Zas(Id_Zas)
);
```

- **Id_Site (INTEGER PRIMARY KEY) :** Identifiant unique du site, agissant comme clé primaire.
- **Code_site (TEXT UNIQUE) :** Code identifiant le site, avec une contrainte d'unicité.
- **Id_Zas (INTEGER) :** Clé étrangère liée à la table Zas (FOREIGN KEY(Id_Zas) REFERENCES Zas(Id_Zas)), assurant une liaison entre les sites et les ZAS.
- Nom_site (TEXT UNIQUE) : Nom du site.
- **Type_impl (TEXT) :** Type d'implantation du site.

Cette table enregistre les informations spécifiques à chaque site, y compris son identifiant unique (Id_Site), un code (Code_site), une référence à la ZAS correspondante (Id_Zas), le nom du site (Nom_site), et le type d'implantation (Type_impl).

La table Mesure:

```
CREATE TABLE Mesure (
   Id_site INTEGER,
```

```
    ID_Polluant INTEGER,
    Date_Deb DATE,
    Date_fin DATE,
    Discriminant TEXT,
    Type_valeur INTEGER,
    valeur_brute NUMERIC,
    Unit INTEGER,
    taux_saisi INTEGER,
    couverture_temp INTEGER,
    Validite INTEGER,
    FOREIGN KEY(Id_site) REFERENCES Site(Id_site),
    FOREIGN KEY(Id_Polluant) REFERENCES Polluant(Id_Polluant)
);
```

- **Id_site (INTEGER) :** Clé étrangère liée à la table Site (FOREIGN KEY(Id_site) REFERENCES Site(Id_Site)), établissant une connexion entre les mesures et les sites.
- **ID_Polluant (INTEGER) :** Clé étrangère liée à la table Polluant (FOREIGN KEY(ID_Polluant) REFERENCES Polluant(Id_Polluant)), établissant une relation entre les mesures et les polluants.
- **Date_Deb (DATE) :** Date de début de la mesure.
- **Date_fin (DATE) :** Date de fin de la mesure.
- **Discriminant (TEXT) :** Information discriminante.
- Type_valeur (INTEGER) : Type de valeur.
- **valeur_brute (NUMERIC) :** Valeur brute de la mesure.
- Unit (INTEGER) : Unité de mesure.
- taux_saisi (INTEGER) : Taux de saisie.
- couverture_temp (INTEGER) : Couverture temporelle.
- **Validite (INTEGER) :** Validité de la mesure.

Cette table enregistre les mesures de polluants effectuées sur chaque site. Les clés étrangères (Id_site et ID_Polluant) lient cette table aux tables Site et Polluant respectivement.

La table Polluant:

```
CREATE TABLE Polluant (
    Id_Polluant INTEGER PRIMARY KEY,
    nom_polluant TEXT UNIQUE,
    Type_influ TEXT
);
```

- **Id_Polluant (INTEGER PRIMARY KEY) :** Identifiant unique du polluant, agissant comme clé primaire.
- **nom_polluant (TEXT UNIQUE) :** Nom du polluant, avec une contrainte d'unicité.
- **Type_influ (TEXT) :** Type d'influence du polluant.

Cette table enregistre les informations sur les différents polluants, tels que leur identifiant unique (Id_Polluant), leur nom (nom_polluant), et leur type d'influence (Type_influ).

En résumé, ces tables interconnectées établissent une structure de base de données complète, répondant à un schéma défini pour stocker de manière organisée et interdépendante les informations relatives aux organismes, aux Zones Administratives de Surveillance (ZAS), aux sites de mesure, aux mesures de polluants et aux polluants eux-mêmes. Cette architecture bien pensée assure une gestion efficace et cohérente des données tout au long du processus du projet

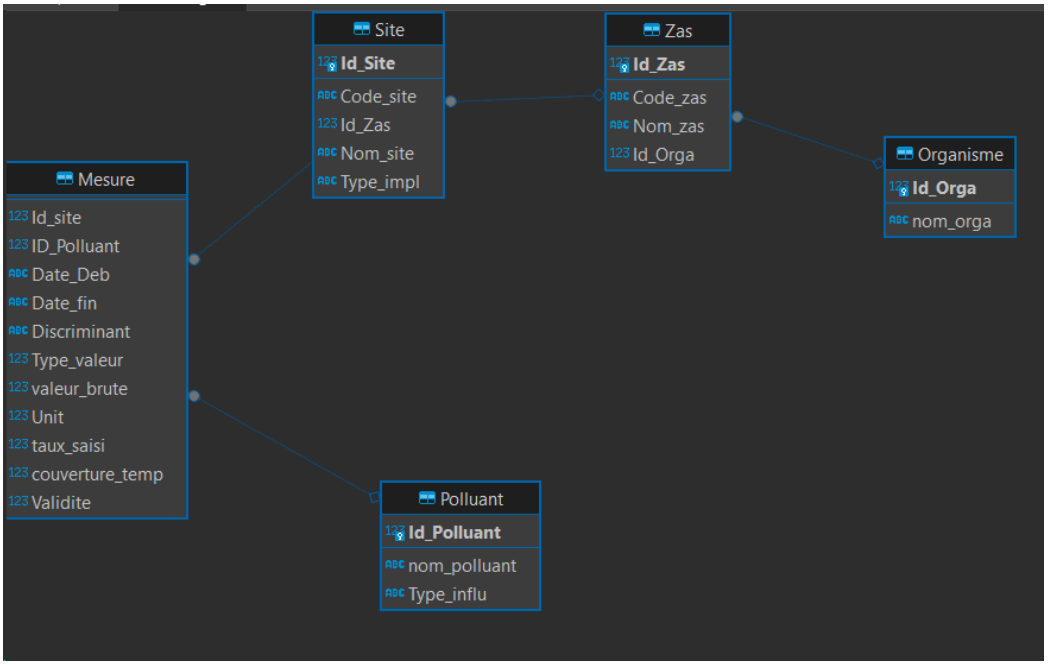
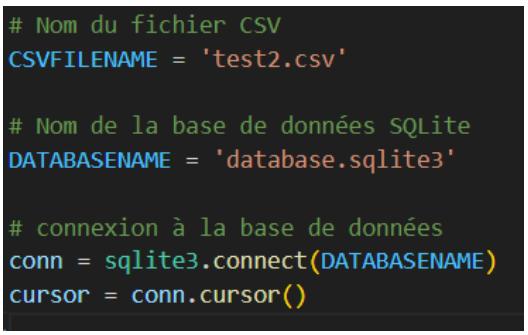


Schéma final de la base de données.

Charger des données dans la base et synchronisation entre le script et DBeaver

Afin de garantir une accessibilité aisée et une utilisation efficiente des données, nous avions développé un script en langage Python qui automatise le chargement de données depuis un fichier CSV, intégrant ainsi un processus efficace d'alimentation de la base de données. Ce dernier constitue une étape cruciale dans la création d'une interface utilisateur intuitive et informative. La base de données contient ainsi les mesures provenant de sites de surveillance de la qualité de l'air répartis à travers le pays, fournissant ainsi une perspective globale des niveaux de polluants. Ce script facilite non seulement la mise à jour régulière des données, mais contribue également à la mise en œuvre d'une plateforme dynamique et réactive pour l'exploration et l'analyse des paramètres environnementaux cruciaux. Ce rapport détaillera le processus de développement du script Python ainsi que les fonctionnalités clés de l'interface web résultante.



```python
# Nom du fichier CSV
CSVFILENAME = 'test2.csv'

# Nom de la base de données SQLite
DATABASENAME = 'database.sqlite3'

# connexion à la base de données
conn = sqlite3.connect(DATABASENAME)
cursor = conn.cursor()
```

Au début du script Python, plusieurs éléments importants sont définis pour faciliter l'intégration des données de mesure de la pollution de l'air. Nous précisons d'abord le nom du fichier CSV contenant les données à l'aide de la variable **"CSVFILENAME".** Ensuite, une base de données SQLite est définie avec le nom **« DATABASENAME ».** Ces deux éléments sont essentiels pour lire les fichiers CSV et stocker les données dans une structure de base de données relationnelle.

Le script utilise ensuite le module SQLite intégré de Python pour se connecter à la base de données SQLite. La variable **« conn »** représente une connexion à la base de données et le **« curseur »** est créé pour interagir avec la base de données en exécutant des requêtes SQL. Cette étape préparatoire est importante pour permettre un transfert efficace des données du fichier CSV vers la base de données et jette les bases de l'interaction future de l'interface Web avec ces données d'environnement.

```
# Exécution de la requête pour récupérer les noms des tables
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")

# Récupération des résultats de la requête
tables = cursor.fetchall()

# Supprimer

print('Vider la base de données ? (1: Oui / Autre: Non)')
n = int(input())

if n == 1:
    cursor.execute('DELETE FROM Mesure;')
    cursor.execute('DELETE FROM Organisme;')
    cursor.execute('DELETE FROM Polluant;')
    cursor.execute('DELETE FROM Site;')
    cursor.execute('DELETE FROM Zas;')
```

Dans cette section du script, une requête SQL est exécutée pour récupérer les noms des tables présentes dans la base de données SQLite. La requête utilise la commande ***"SELECT name FROM sqlite_master WHERE type='table';"*** pour extraire ces informations. Les résultats de la requête sont ensuite stockés dans la variable **« tables »** à l'aide de la méthode **« fetchall() ».**

Par la suite, le script propose une interaction avec l'utilisateur, lui demandant s'il souhaite vider la base de données. La question est affichée avec la commande « **print »,** suivie d'une demande d'entrée numérique (1 pour Oui, tout autre chiffre pour Non). Si l'utilisateur choisit de vider la base de données (choix 1), des requêtes SQL supplémentaires sont exécutées pour supprimer toutes les données présentes dans chacune des tables spécifiées (Mesure, Organisme, Polluant, Site, Zas). Cette fonctionnalité permet de préparer la base de données pour une mise à jour avec de nouvelles données provenant du

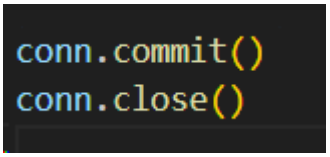
fichier CSV, assurant ainsi un environnement de travail propre et actualisé.

```
# Chargement des données depuis le fichier CSV
with open(CSVFILENAME, 'r') as csv_file:
    # La première ligne est passée (headers)
    next(csv_file)

    # Lecture du fichier CSV et insertion des données dans les tables
    csv_data = csv.reader(csv_file, delimiter=";")
    for row in csv_data:
        # Insertion ou récupération de l'id de l'organisme
        cursor.execute('INSERT OR IGNORE INTO Organisme (nom_orga) VALUES (?)', (row[2],))
        cursor.execute('SELECT Id_Orga FROM Organisme WHERE nom_orga = ?', (row[2],))
        Id_Orga = cursor.fetchone()[0]

        # Insertion ou récupération de l'id du Zas
        cursor.execute('INSERT OR IGNORE INTO Zas ( Code_zas, Nom_zas, Id_Orga) VALUES (?, ?, ?)', (row[3],row[4],Id_Orga))
        cursor.execute('SELECT Id_Zas FROM Zas WHERE code_zas = ?', (row[3],))
        Id_Zas = cursor.fetchone()[0]

        # Insertion ou récupération du l'id du Site
        cursor.execute('INSERT OR IGNORE INTO Site (Code_site, Nom_Site, Type_Impl, Id_Zas) VALUES (?, ?, ?, ?)', (row[5],row[6], row[7],Id_Zas))
        cursor.execute('SELECT Id_Site FROM SITE WHERE Nom_Site = ?', (row[6],))
        Id_Site = cursor.fetchone()[0]

        # Insertion ou Récupération de l'id de Pollution
        cursor.execute('INSERT OR IGNORE INTO Polluant (nom_polluant, type_influ) VALUES (?, ?)', (row[8], row[9]))
        cursor.execute('SELECT Id_Polluant FROM Polluant WHERE nom_polluant = ?', (row[8],))
        Id_Polluant = cursor.fetchone()[0]

        # Insertion ou récupération de l'id de la Mesure
        cursor.execute('INSERT OR IGNORE INTO Mesure ( Id_Site, Id_Polluant, Date_Deb, Date_fin, Discriminant, Type_Valeur, Valeur_Brute, Unit, Tau
```

Dans cette section du script, le processus de chargement des données depuis le fichier CSV vers la base de données SQLite est mis en œuvre. Le code utilise une boucle **« for »** pour parcourir chaque ligne du fichier CSV après avoir omis la première ligne contenant les en-têtes. Pour chaque ligne, les données sont extraites et insérées dans les tables correspondantes de la base de données.

Le script commence par gérer les informations liées à l'organisme de mesure. Il insère ou récupère l'ID de l'organisme à partir de la table ***"Organisme".*** Ensuite, il fait de même pour le Zas (Zone d'Action Simplifiée) et le site de mesure, récupérant ou insérant les ID nécessaires dans les tables ***"Zas"*** et ***"Site".*** De manière similaire, il gère l'information sur le polluant, insérant ou récupérant l'ID du polluant depuis la table "Polluant".

Enfin, le script traite les données de mesure proprement dites. Il insère ou récupère l'ID du site, de l'organisme, du polluant, puis effectue l'insertion dans la table ***"Mesure".*** Ce processus assure l'intégrité des relations entre les différentes entités, garantissant une structure de base de données cohérente. L'utilisation de la clause ***"INSERT OR IGNORE"*** évite les duplications d'informations déjà présentes dans la base de données. En résumé, ce script automatisé facilite le chargement des données du fichier CSV dans la base de données, créant ainsi une fondation solide pour l'interface web dédiée aux mesures de pollution atmosphérique en France.



La commande conn.commit() valide toutes les modifications apportées à la base de données depuis le début de la transaction. Cela inclut les insertions, mises à jour et suppressions effectuées à l'aide de diverses requêtes SQL. L'exécution de cette commande enregistre définitivement toutes les modifications apportées à la base de données. La commande "conn.close()" ferme alors la connexion à la base de données. La fermeture des connexions est importante pour libérer des ressources et éviter les problèmes potentiels liés aux connexions persistantes. Lorsqu'une connexion est fermée, toutes les opérations de base de données associées à cette connexion ne peuvent pas être effectuées tant qu'une nouvelle connexion n'est pas établie.

En résumé, ces deux commandes sont importantes pour garantir l'intégrité des données et libérer des ressources après le traitement des opérations de base de données.

Test et vérification

Afin de garantir la cohérence et l'intégrité de la base de données, une série de requêtes SQL a été exécutée sous l'environnement Collaboratory (Collab), permettant ainsi de vérifier la structure, les relations et la qualité des données enregistrées dans le système.

```
[2] %load_ext sql

    %sql sqlite:///database.sqlite3
```

```
[4] %%sql
    SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite%'
```

```
 * sqlite:///database.sqlite3
Done.
   name
Organisme
Zas
Site
Polluant
Mesure
```

Nous avons ainsi chargé la base de donnée (1ère Bulle)

La première requête est destinée à récupérer les noms des tables présentes dans la base de données SQLite, à l'exception des tables système commençant par 'sqlite%'.

Nous récupérons ainsi l'id du site de Vitry pour ensuite récupérer les informations de mesures du 1er Janvier 2023.

```
%%sql
select * from site
where Nom_Site LIKE '%Vitry%'
```

```
 * sqlite:///database.sqlite3
Done.
Id_Site Code_site Id_Zas  Nom_site       Type_impl
60      FR04034   7       VITRY-SUR-SEINE Urbaine
```

```
[11] %%sql
SELECT Nom_Site, Date_Deb, valeur_brute FROM Mesure M, Site S
WHERE S.Id_Site = M.Id_Site
  AND S.Id_Site = 60
```

```
VITRY-SUR-SEINE 2023/01/01 00:00:00 55.1
VITRY-SUR-SEINE 2023/01/01 01:00:00 53.65
VITRY-SUR-SEINE 2023/01/01 02:00:00 51.7
VITRY-SUR-SEINE 2023/01/01 03:00:00 52.5
VITRY-SUR-SEINE 2023/01/01 04:00:00 54.05
VITRY-SUR-SEINE 2023/01/01 05:00:00 54.2
```