EQUIPE 5

LIVRABLE 2 : DONNÉES ET MODÉLISATION

KERGOAT-DUBART EWEN

GLAIROT VALENTIN

FERAT ALEXANDRE

LEGUERNEY CAMILLE

VERGER VINCENT

SOMMAIRE :

INTRODUCTION :

Suite au dernier rapport du GIEC, le ministère de l'Écologie a chargé notre entreprise DATA-X de développer un outil pour stocker et interroger les données publiques sur la qualité de l'air dans les grandes villes de France. Cette initiative vise à centraliser les informations dispersées dans les agences météorologiques locales. En partenariat avec le ministère, DATA-X doit concevoir une base de données relationnelle pour gérer efficacement ces données et répondre aux exigences spécifiées lors de réunions avec les responsables gouvernementaux. Dans cette deuxième réunion, nous allons donc fournir au ministère de l'écologie la base de données finie, peuplée avec les requêtes permettant de l'interroger.

CRÉATION DE LA BASE DE DONNÉES :

Pour créer la base de données demandée, nous avons utilisé le MPD présenté lors de la réunion précédente. En effet, nous avons exécuté le code (MPD) dans MySQL ce qui nous a permis de créer nos tables, clés primaires, clés étrangères, etc.

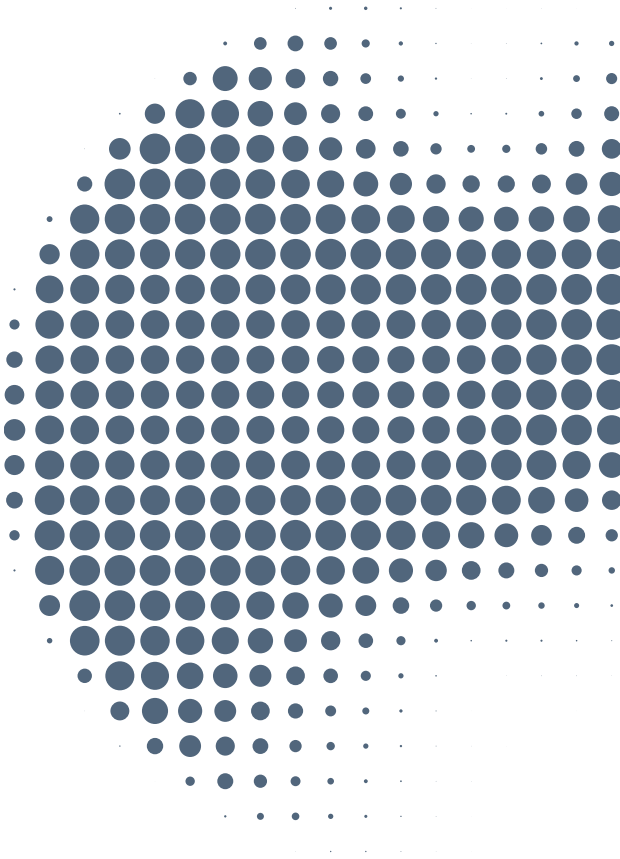
<- Base de données implémentée dans MySQL

On peut voir que notre base de données a bien été créée dans MySQL, on peut voir ici toutes les tables qui ont été créées en important donc un fichier CSV par table.

PEUPLEMENT DE LA BASE DE DONNÉE :

Pour le peuplement de notre base de données, nous avons dû faire un choix. Nous avons choisi de créer des scripts Python de génération aléatoire pour compléter notre base de données. En effet, nos scripts nous ont permis de générer, par exemple, des noms, prénoms, dates de naissance, dates d'embauche aléatoirement et surtout rapidement.

```
import pandas as pd
from faker import Faker
import random

# Initialisation de Faker
fake = Faker('fr_FR')

# Génération des données
num_rows = 200
data = {
    'Numero_rue': [random.randint(1, 50) for _ in range(num_rows)],
    'Nom_rue': [fake.street_name() for _ in range(num_rows)],
    'Code_postal': [fake.postcode() for _ in range(num_rows)],
    'Nom_ville': [fake.city() for _ in range(num_rows)]
}

# Création du DataFrame
df = pd.DataFrame(data)

# Export vers un fichier Excel
excel_filename = 'coordonnees_francaises.xlsx'
df.to_excel(excel_filename, index=False)

print(f"Données exportées vers {excel_filename}")
```



Cet exemple de code nous a permis de générer aléatoirement des adresses pour nos employés, par exemple. Le code nous renvoie dans un fichier CSV directement nos adresses, il ne nous reste plus qu'à les importer dans MySQL.

REQUÊTE SQL :

Nous avons 12 requêtes à fournir, qui vont interroger notre base de données.
Nous allons d'abord expliquer le fonctionnement et la syntaxe des requêtes dans leur globalité, puis détailler rapidement chaque requête.
Une requête se sépare en plusieurs clauses, la clause “**SELECT**” est utilisée pour spécifier les colonnes que l’on souhaite récupérer dans le résultat de la requête, la clause “**FROM**” est utilisée pour spécifier la table ou les tables à partir desquelles l’on souhaite récupérer les données. La clause “**JOIN**” est utilisée pour combiner des lignes de deux ou plusieurs tables basées sur une colonne commune entre elles. La clause “**WHERE**” est utilisée pour filtrer les résultats de la requête en spécifiant une condition. La clause “**ORDER BY**” est utilisée pour trier les résultats d'une requête SQL selon une ou plusieurs colonnes spécifiées.
À partir de cette liste de clauses non exhaustive, nous pouvons facilement interroger notre base de données, nous allons donc expliquer chaque requête demandée.

```
SELECT Nom_agence
FROM Agence;
```

Cette requête sélectionne la colonne appelée "Nom_agence" de la table "Agence" et renvoie tous les enregistrements qui se trouvent dans cette colonne.

```
SELECT e.Nom_personnel, e.Prenom_personnel
FROM Employe e
JOIN Travail t ON e.id_personnel = t.id_personnel
JOIN Agence a ON t.id_agence = a.id_agence
JOIN Adresse adr ON a.id_adresse = adr.id_adresse
JOIN Region r ON adr.id_region = r.id_region
WHERE a.Nom_agence = 'Agence_Bordeaux';
```

Cette requête SQL extrait les noms et prénoms des employés travaillant dans l'agence nommée "Agence_Bordeaux". Elle effectue une série de jointures entre les tables "Employé", "Travail", "Agence", "Adresse" et "Région" afin de lier les données. On sélectionne les colonnes appropriées de la table "Employé" et utilise une condition dans la clause WHERE pour filtrer les résultats en fonction du nom de l'agence.

```
SELECT COUNT(*) AS Nombre_de_capteurs_deployes
FROM Capteur;
```

Cette requête SQL compte le nombre total de capteurs déployés dans un système en récupérant toutes les lignes de la table "Capteur" et en comptant leur nombre. Le résultat est renommé en tant que "Nombre_de_capteurs_deployes" pour une meilleure lisibilité.

```
SELECT *
FROM Rapport
WHERE date_publication BETWEEN '2018-01-01' AND '2022-12-31';
```

Cette requête SQL extrait toutes les colonnes de la table "Rapport" où la date de publication se situe dans la plage spécifiée, c'est-à-dire entre le 1er janvier 2018 et le 31 décembre 2022.

```
SELECT R.Nom_region, SUM(D.valeur_donnee) AS Total_emissions
FROM Donnee D
INNER JOIN Capteur C ON D.id_capteur = C.id_capteur
INNER JOIN Adresse A ON C.id_adresse = A.id_adresse
INNER JOIN Region R ON A.id_region = R.id_region
INNER JOIN Identifier I ON C.id_capteur = I.id_capteur
INNER JOIN Gaz G ON I.id_gaz = G.id_gaz
WHERE YEAR(D.Date_donnee) = 2020
GROUP BY R.Nom_region;
```

Cette requête SQL calcule le total des émissions pour chaque région en 2020. Pour cela, elle effectue plusieurs jointures entre les tables "Donnée", "Capteur", "Adresse", "Région", "Identifier" et "Gaz". Ensuite, elle filtre les données pour ne prendre en compte que celles de l'année 2020, puis regroupe les résultats par nom de région.

```
SELECT C.secteur_activite, SUM(D.valeur_donnee) AS
Total_emissions
FROM Donnee D
INNER JOIN Capteur C ON D.id_capteur = C.id_capteur
INNER JOIN Adresse A ON C.id_adresse = A.id_adresse
INNER JOIN Region R ON A.id_region = R.id_region
WHERE R.Nom_region = 'Ile de France'
GROUP BY C.secteur_activite
ORDER BY Total_emissions DESC;
```

Cette requête SQL calcule le total des émissions par secteur d'activité pour la région "Île-de-France". Elle réalise des jointures entre les tables "Donnée", "Capteur", "Adresse" et "Région". Ensuite, elle filtre les données pour ne considérer que celles de la région "Île-de-France", puis elle groupe les résultats par secteur d'activité. Pour finir, les résultats sont triés par ordre décroissant (DESC) en fonction du total des émissions par secteur d'activité.

```
SELECT r.*
FROM Rapport r
JOIN Contenir c ON r.id_rapport = c.id_rapport
JOIN Donnee d ON c.id_donnee = d.id_donnee
JOIN Capteur cp ON d.id_capteur = cp.id_capteur
JOIN Identifier i ON cp.id_capteur = i.id_capteur
JOIN Gaz g ON i.id_gaz = g.id_gaz
WHERE g.Type_gaz = 'NH3'
ORDER BY r.date_publication;
```

Cette requête SQL extrait toutes les colonnes de la table "Rapport" liées aux rapports concernant le gaz NH3. Elle réalise plusieurs jointures avec les tables "Contenir", "Donnée", "Capteur", "Identifier" et "Gaz". Elle filtre ensuite les résultats pour inclure uniquement les rapports concernant le gaz NH3. Pour finir, les résultats sont triés par date de publication des rapports, du plus ancien au plus récent.

```
SELECT DISTINCT Employe.Nom_personnel,
Employe.Prenom_personnel
FROM Employe
JOIN Capteur ON Employe.id_personnel = Capteur.id_personnel
JOIN Identifier ON Capteur.id_capteur = Identifier.id_capteur
JOIN Gaz ON Identifier.id_gaz = Gaz.id_gaz
WHERE Gaz.Type_gaz IN ('NH3', 'CO2 non bio','CH4');
```

Cette requête SQL extrait les noms et prénoms des employés associés à des capteurs mesurant les gaz NH3, CO2 non bio et CH4 qui sont des gaz polluants acidificateurs. Elle effectue des jointures entre les tables "Employe", "Capteur", "Identifier" et "Gaz" pour ensuite filtrer les résultats pour inclure uniquement les employés associés à ces types de gaz spécifiques.

```
SELECT G.Type_gaz, SUM(D.valeur_donnee) AS
Somme_emissions_ppm
FROM Donnee D
INNER JOIN Capteur C ON D.id_capteur = C.id_capteur
INNER JOIN Adresse A ON C.id_adresse = A.id_adresse
INNER JOIN Region R ON A.id_region = R.id_region
INNER JOIN Identifier I ON C.id_capteur = I.id_capteur
INNER JOIN Gaz G ON I.id_gaz = G.id_gaz
WHERE R.Nom_region = 'Ile de France'
AND YEAR(D.Date_donnee) = 2020
GROUP BY G.Type_gaz;
```

Cette requête SQL calcule la somme des émissions pour chaque type de gaz enregistré en ppm (parties par million) pour la région "Île-de-France" en 2020. Pour cela, elle effectue des jointures entre les tables "Donnée", "Capteur", "Adresse", "Région", "Identifiant" et "Gaz". Elle filtre ensuite les données pour ne considérer que celles de la région "Île-de-France" en 2020, puis elle groupe les résultats par type de gaz.

```
SELECT E.Nom_personnel, E.Prenom_personnel, COUNT(R.id_rapport) AS
Nombre_de_rapports_ecrits, DATEDIFF(CURDATE(), E.Date_naissance) AS
Anciennete_dans_le_poste, COUNT(R.id_rapport) / DATEDIFF(CURDATE(),
E.Date_naissance) AS Taux_de_productivite
FROM Employe E
JOIN Ecriture EC ON E.id_personnel = EC.id_personnel
JOIN Rapport R ON EC.id_rapport = R.id_rapport
JOIN Travail T ON E.id_personnel = T.id_personnel
JOIN Agence A ON T.id_agence = A.id_agence
JOIN Adresse Ad ON A.id_adresse = Ad.id_adresse
JOIN Region Re ON Ad.id_region = Re.id_region
WHERE A.Nom_agence = 'Agence_Toulouse'
GROUP BY E.id_personnel
ORDER BY Taux_de_productivite DESC;
```

Cette requête SQL fournit le taux de productivité des employés travaillant à l'agence "Agence_Toulouse". Elle sélectionne le nom, le prénom de chaque employé, compte le nombre de rapports écrits par chaque employé, calcule leur ancienneté dans le poste en jours, puis détermine leur taux de productivité en divisant le nombre de rapports écrits par leur ancienneté. Pour cela, elle réalise plusieurs jointures entre les tables "Employe", "Ecriture", "Rapport", "Travail", "Agence", "Adresse" et "Region". Elle filtre ensuite les résultats pour ne considérer que les employés de l'agence "Agence_Toulouse", puis elle groupe les résultats par employé. Pour finir, elle trie les résultats par ordre décroissant de taux de productivité, permettant d'identifier les employés les plus productifs en premier.

```
DELIMITER //
CREATE PROCEDURE getStudentInfo(IN s_name VARCHAR(64))
BEGIN
SELECT R.id_rapport, R.Nom_rapport, R.date_publication,
R.Date_de_modification
FROM Rapport R
JOIN Contenir C ON R.id_rapport = C.id_rapport
JOIN Donnee D ON C.id_donnee = D.id_donnee
JOIN Identifier I ON D.id_capteur = I.id_capteur
JOIN Gaz G ON I.id_gaz = G.id_gaz
WHERE G.Type_gaz = s_name;
END//
call getStudentInfo('CH4');
```

Cette requête SQL crée une procédure stockée appelée "getStudentInfo" qui prend un paramètre d'entrée "s_name" de type VARCHAR(64). Cette procédure stockée récupère les informations sur les rapports concernant un type de gaz spécifié et les renvoie. On crée ensuite une procédure stockée, et nous déterminons le délimiteur DELIMITER //. À l'intérieur du bloc de la procédure stockée, une requête SELECT est utilisée pour récupérer les informations des rapports qui correspondent au type de gaz spécifié. La requête utilise plusieurs jointures pour lier les tables "Rapport", "Contenir", "Donnée", "Identifier" et "Gaz" afin d'obtenir les informations nécessaires.
La clause WHERE filtre les résultats pour ne retourner que les rapports concernant le type de gaz spécifié par le paramètre d'entrée "s_name". Pour finir, la procédure stockée est appelée avec l'instruction call getStudentInfo('CH4'); où 'CH4' est le type de gaz pour lequel nous voulons récupérer les informations sur les rapports.

```
SELECT R.Nom_region
FROM Region R
JOIN ( SELECT Ad.id_region, COUNT(DISTINCT C.id_capteur) AS nb_capteurs,
COUNT(DISTINCT A.id_agence) AS nb_agences
    FROM Adresse Ad
    LEFT JOIN Agence A ON Ad.id_adresse = A.id_adresse
    LEFT JOIN Capteur C ON Ad.id_adresse = C.id_adresse
    GROUP BY Ad.id_region) AS T ON R.id_region = T.id_region
WHERE
  T.nb_capteurs > T.nb_agences;
```

Cette requête SQL extrait les noms des régions où le nombre de capteurs est supérieur au nombre d'agences. Elle utilise une sous-requête pour calculer le nombre de capteurs et le nombre d'agences pour chaque région, puis joint ces résultats avec la table "Region". Elle filtre ensuite les résultats pour inclure uniquement les régions où le nombre de capteurs est plus grand que le nombre d'agences.

CONCLUSION :

Notre société a correctement établi les différents points de développement, tels que la création de la base de données, le peuplement de celle-ci ainsi que les requêtes SQL demandées. Certaines difficultés ont été rencontrées lors du peuplement avec les types de dates "JJ-MM-AAAA" ou encore la concordance des clés primaires / étrangères.

Certains points de notre base de données peuvent être améliorés / optimisés, comme une interface de peuplement et d'interrogation de la base de données pour faciliter l'accès de notre client à l'outil développé. Notre entreprise a été ravie de contribuer et de développer cette base de données. L'équipe reste à votre disposition pour tout complément d'information concernant votre projet.