# Data Quality

Membres: Lucas Chipan, Belkis Coskun, Lyes Boumrah, Victory Dimbou Mbanzilla
ingénierie des données I1.

## Exercice 1 : Configuration et prise en main

Dans le cadre de ce premier exercice, nous allons aborder la configuration de Soda Core et tester son bon fonctionnement en effectuant des vérifications simples sur une base de données PostgreSQL.

### 1. Création du fichier de configuration (configuration.yml)

La première étape consiste à créer un fichier de configuration YAML permettant à Soda Core de se connecter à la base de données PostgreSQL.

Voici à quoi ressemble le contenu du fichier `configuration.yml` :

```yaml
data_source nyc_warehouse:
  type: postgres
  connection:
        host: localhost
        port: 15432
        username: postgres
        password: admin
        database: nyc_warehouse
        schema: public
```

Une fois ce fichier créé, il est nécessaire de tester la connexion à la base de données avec la commande suivante :

```bash
soda test-connection -d nyc_warehouse -c configuration.yml
```

Suite à une configuration correcte, voici le message qui s'est affiché:

```bash
Soda Core 3.4.1
Successfully connected to 'nyc_warehouse'.
```

Connection 'nyc_warehouse' is valid.
```

```
(data_archi) C:\Users\lucas\OneDrive\Desktop\EPSI\Architecture_des_données\TP1\Part 1\Data _QualityTP>soda test-connection -d nyc_warehouse -c configuration.yml
[04:26:48] Soda Core 3.4.4
Successfully connected to 'nyc_warehouse'.
Connection 'nyc_warehouse' is valid.
```

## 2. Création et test du fichier check.yaml

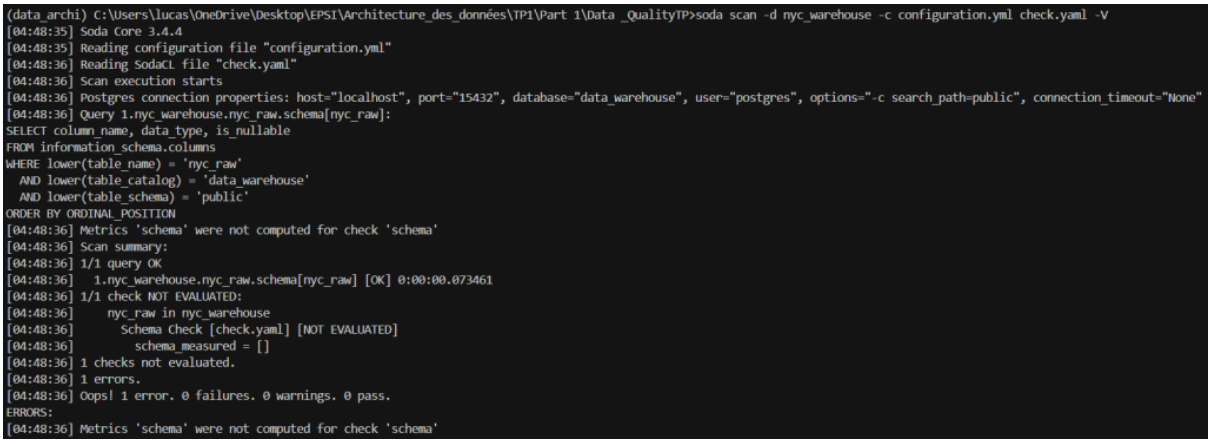
Ensuite, nous allons créer un fichier de vérification `check.yaml` qui définira les règles que Soda Core utilisera pour analyser les données dans la table `nyc_raw`.
Nous allons spécifier des vérifications liées au schéma de la table, en particulier pour les colonnes requises et interdites. Voici le contenu du fichier `check.yaml` :

```yaml
checks for nyc_raw:
  - schema:
        warn:
        when required column missing: [ column_name ]
        fail:
        when forbidden column present: [ column_name, column_name2 ]
```

Une fois le fichier `check.yaml` créé, nous pouvons tester les vérifications avec la commande suivante :
```bash
soda scan -d nyc_warehouse -c configuration.yml check.yaml
```

```
(data_archi) C:\Users\lucas\OneDrive\Desktop\EPSI\Architecture_des_données\TP1\Part 1\Data _QualityTP>soda scan -d nyc_warehouse -c configuration.yml check.yaml
[04:27:31] Soda Core 3.4.4
[04:27:32] Metrics 'schema' were not computed for check 'schema'
[04:27:32] Scan summary:
[04:27:32] 1/1 check NOT EVALUATED:
[04:27:32]     nyc_raw in nyc_warehouse
[04:27:32]       Schema Check [NOT EVALUATED]
[04:27:32]         schema_measured = []
[04:27:32] 1 checks not evaluated.
[04:27:32] 1 errors.
[04:27:32] Oops! 1 error. 0 failures. 0 warnings. 0 pass.
ERRORS:
[04:27:32] Metrics 'schema' were not computed for check 'schema'
```

Ensuite, en ajoutant l'option `-V` pour obtenir un mode détaillé (verbose), la commande devient :
```




La commande avec le -V permet quant à elle de fournir plus de détails sur le processus d'exécution comme la lecture des fichiers utilisés sous forme de confirmation; les détails de la connexion de PostgreSQL avec l'hôte, port, base de données, user, options …). Cette commande est plus utile pour diagnostiquer des problèmes ou analyser

### 3. Ajout d'une règle row_count

Pour cette étape, nous allons modifier le fichier `check.yaml` afin de définir une règle concernant le nombre de lignes dans la table `nyc_raw`.
Nous allons spécifier que :
- Si le nombre de lignes est supérieur à 90, un avertissement doit être généré.
- Si le nombre de lignes est égal à 0, une erreur doit être générée.

Voici à quoi ressemble la nouvelle règle `row_count` dans le fichier `check.yaml` :

```yaml
checks for nyc_raw:
  - row_count:
        warn: when > 90
        fail: when = 0
```


- Il y a eu un warning car la table contient plus de 90 lignes. Il n'y a pas eu d'erreur parce que la table n'est pas vide, elle contient 26 388 179 lignes

## Exercice 2 : Mes premiers checks

Explications des Règles de Qualité des Données

**1. Vérification de l'intégrité des identifiants**

Cette règle s'assure que tous les identifiants comme VendorID et RateCodeID sont présents et correctement formatés selon les spécifications. VendorID doit être 1 ou 2, indiquant le fournisseur du service, et RateCodeID doit être un nombre entre 1 et 6, correspondant aux différents tarifs appliqués lors des trajets.

**2. Vérification de la validité des timestamps**

Les champs tpep_pickup_datetime et tpep_dropoff_datetime doivent contenir des timestamps valides, où la date et l'heure de fin de course sont postérieures à celles du début.

**3. Vérifications des montants et des frais**

Cette règle assure que tous les montants financiers liés au trajet, tels que Fare_amount, Tip_amount, et Total_amount, sont non seulement présents mais aussi positifs. Le Total_amount doit correspondre à la somme de tous les frais applicables.

**4. Validation des conditions spéciales**

Le champ Store_and_fwd_flag, qui indique si les données du trajet ont été stockées avant transmission au serveur, doit être soit 'Y' (oui) soit 'N' (non). Payment_type doit aussi être compris entre 1 et 6, reflétant le mode de paiement utilisé.

**5. Cohérence géographique**

Les identifiants PULocationID et DOLocationID, qui représentent respectivement les zones de prise en charge et de dépose des passagers, doivent correspondre à des valeurs valides de la table des zones de taxi.