# Documentation Technique du Test

## Table des matières

1. [Structure du Projet](#structure-du-projet)
2. [Schéma de Base de Données](#schéma-de-base-de-données)
3. [Scripts Python](#scripts-python)
4. [Configuration Docker](#configuration-docker)
5. [Commandes d’Exécution](#commandes-dexécution)
6. Problèmes rencontrés

## Structure du Projet

dt\_eng\_test/

├── data/

│ ├── people.csv

│ ├── places.csv

│ ├── sample\_output.json

│ ├── total\_births\_by\_country

│ └── summary\_output.json

│

├── docker-compose.yml

├── example\_schema.sql

│

├── load-csv/

│ ├── Dockerfile

│ ├── load\_to\_db.py

│ └── requirements.txt

│

└── summary-output/

├── Dockerfile

├── summary\_output.py

└── requirements.txt

Le fichier example\_schema.sql définit trois tables :

**Suppression des tables et de leur contenu si elles existent :**

drop table if exists people;

drop table if exists places;

drop table if exists examples;

Schema de la table de demonstration examples:

create table `examples` (

`id` int not null auto\_increment,

`name` varchar(80) default null,

primary key (`id`)

);

**Schema de la table people :**

create table `people`(

`id` int not null auto\_increment,

`given\_name`varchar(80) not null,

`family\_name` varchar(80) not null,

`date\_of\_birth` DATE,

`place\_of\_birth` varchar(100),

PRIMARY KEY(`id`),

UNIQUE KEY unique\_name\_dob (given\_name, family\_name, date\_of\_birth)

);

**Schema de la table Places :**

create table `places`(

`id` int not null auto\_increment,

`city` varchar(80) not null,

`county` varchar(80) not null,

`country` varchar(80) not null,

PRIMARY KEY(`id`),

UNIQUE KEY unique\_ccc (city, county, country)

### )

### Points clés du schéma

* La table people utilise une contrainte d’unicité sur la combinaison (given\_name, family\_name, date\_of\_birth)
* La table places utilise une contrainte d’unicité sur la combinaison (city, county, country)
* Toutes les tables utilisent l’auto-incrémentation pour les clés primaires

## Scripts Python

**Script de comptage des naissances dans chaque pays depuis les fichiers csv seulement :**

Dans ce script j’ai utilisé la bibliothèque pandas pour charger et manipuler des données de plusieurs fichiers CSV, puis produit un fichier json qui contient le nombre de naissance par pays.

### Fonctionnement du script

1. **Chargement des fichiers CSV en DataFrames** :
   * Le script commence par charger deux fichiers CSV, people.csv et places.csv, situés dans le répertoire data. La fonction pd.read\_csv() de pandas est utilisée pour lire chaque fichier et convertir leur contenu en DataFrames nommés people\_df et places\_df.
   * Ces DataFrames permettent de structurer les données de manière tabulaire, facilitant les opérations de jointure et d'agrégation.
2. **Jointure des données** :
   * Ensuite, j’effectue une jointure entre les deux DataFrames (people\_df et places\_df) en utilisant les colonnes city et place\_of\_birth comme clés. La méthode merge() de pandas permet de combiner les informations des deux tables en créant un DataFrame unifié merged\_df.
   * La jointure est de type left join, ce qui signifie que toutes les entrées de people\_df seront conservées, même si elles n’ont pas de correspondance dans places\_df.
3. **Agrégation des naissances par pays** :
   * J’utilise ensuite groupby() pour regrouper les données par pays (country) et calculer le nombre total de naissances pour chaque pays. La fonction size() compte le nombre d’occurrences pour chaque pays, et sort\_values(ascending=False) trie les résultats par ordre décroissant de nombre de naissances.
   * Le résultat est converti en dictionnaire avec to\_dict(), produisant une structure clé-valeur où chaque pays est associé au nombre total de naissances.
4. **Enregistrement du résultat dans un fichier JSON** :
   * Le dictionnaire births\_by\_country est ensuite enregistré dans un fichier JSON nommé total\_births\_by\_country.json dans le répertoire data.
   * J’ai utilisé la fonction json.dump() pour écrire les données dans un fichier, avec un format sans espace pour optimiser la taille du fichier. Un message de confirmation est affiché si le fichier est créé avec succès.
   * En cas d’erreur lors de l’écriture, le script capture l'exception IOError et affiche un message d'erreur pour informer l'utilisateur.
5. **Gestion des erreurs** :
   * Le script est encapsulé dans un bloc try-except pour gérer les erreurs éventuelles liées au chargement des fichiers CSV. Si l'un des fichiers est manquant, une exception FileNotFoundError est levée, et un message d’erreur est affiché.

### Script d’Ingestion (load\_to\_db.py)

Ce script utilise SQLAlchemy pour charger les données des fichiers CSV dans la base de données MySQL.

Principales fonctionnalités :

* 5 tentatives de connexion en cas d’échec de la connexion à la base de données avec un intervalle de 5 secondes entre chaque tentative
* Connexion à la base de données MySQL
* Le script est encapsulé dans un bloc try-except pour gérer les erreurs éventuelles liées au à la connexion à la base données et à la récupération des données depuis la base et les éventuels problèmes avec les fichiers csv. Si le script n’arrive pas à se connecter sur la base de données après plusieurs tentatives de connexion, une exception SQLAlchemyError est levée, et un message d’erreur est affiché.
* Lecture des fichiers CSV (people.csv et places.csv)
* Insertion des données dans les tables respectives
* Gestion des erreurs pour les fichiers manquants et les erreurs d’insertion

Exemple de connexion à la base de données :

engine=sqlalchemy.create\_engine("mysql://user\_dteng:cirilgroupt@database/db\_dteng")  
  
Chargement des tables avec SQLAlchemy

Places = sqlalchemy.schema.Table('places', metadata, autoload=True, autoload\_with=engine)  
People = sqlalchemy.schema.Table('people', metadata, autoload=True, autoload\_with=engine)  
  
Insertion des données

connection.execute(People.insert().values(  
 given\_name = row[0],  
 family\_name = row[1],  
 date\_of\_birth = row[2],  
 place\_of\_birth = row[3]  
))

### Script de Génération du Résultat (summary\_output.py)

Ce script génère un fichier JSON contenant le nombre de naissance par pays.

Fonctionnalités principales :

* 5 tentatives de connexion en cas d’échec de la connexion à la base de données avec un intervalle de 5 secondes entre chaque tentative
* Le script est encapsulé dans un bloc try-except pour gérer les erreurs éventuelles liées au à la connexion à la base données et à la récupération des données depuis la base et les éventuels problèmes avec les fichiers csv. Si le script n’arrive pas à se connecter sur la base de données après plusieurs tentatives de connexion, une exception SQLAlchemyError est levée, et un message d’erreur est affiché.
* Création d’un mapping ville → pays
* Comptage des naissances par pays
* Génération d’un fichier JSON avec les résultats

Création du mapping ville → pays

places\_query = connection.execute(sqlalchemy.select([Places.c.city, Places.c.country]))  
for row in places\_query:  
 place\_to\_country[row['city']] = row['country']  
  
Comptage des naissances par pays

people\_query=connection.execute(sqlalchemy.select([People.c.place\_of\_birth]))  
for row in people\_query:  
 place\_of\_birth = row['place\_of\_birth']  
 country\_of\_birth = place\_to\_country.get(place\_of\_birth)  
 if country\_of\_birth:  
 birth\_counts[country\_of\_birth] += 1

## Configuration Docker

### Docker Compose

Il a été nécessaire de créer un réseau docker pour la communication entre les conteneurs en exécutant la commande

Le fichier docker-compose.yml définit quatre services :

1. **database** : Serveur MySQL
   * Image : mysql:8.0
   * Network: réseau docker
   * Exposition du port 3306
   * Configuration des variables d’environnement pour MySQL
2. **example-python** : Service example-python
   * Build depuis le contexte image-python
3. **load-csv** : Service d’ingestion
   * Build depuis le contexte load-csv
   * Network: réseau docker
   * Dépend du service database
   * Monte le volume ./data
4. **summary-output** : Service de génération du résultat
   * Build depuis le contexte summary-output
   * Network: réseau docker
   * Dépend du service database
   * Monte le volume ./data

## Commandes d’Exécution

1. En développant dans un environnement Windows c’est primordial de supprimer les retours à la ligne Windows dans les scripts d’exécution :

sed -i 's/\r$//' load-csv/load\_to\_db.py  
sed -i 's/\r$//' summary-output/summary\_output.py

1. Construction et démarrage des services :

Construction des images  
docker-compose build  
  
Démarrage en arrière-plan  
docker-compose up -d  
  
Arrêt des services  
docker-compose down

1. Vérification des logs :

Les logs de tous les services  
docker-compose logs  
  
Logs du Service spécifique  
docker-compose logs database  
docker-compose logs load-csv  
docker-compose logs summary-output

1. Exécution individuelle des services:

Création des tables dans la base de données

docker compose run --no-TTY database mysql --host=database --user=user\_dteng -- password=cirilgroupt db\_dteng <example\_schema.sql

Chargement des données  
docker-compose run load-csv  
  
Génération du résultat  
docker-compose run summary-output

## Problèmes rencontrés :

Erreur lors du lancement du script de chargement de données/génération de résultat à cause des retours à la ligne Windows.

Solution : exécution de la commande sed -i ‘s/\r$//’ sur chaque fichier afin de les supprimer

Echec de se connecter à la base de données dans un environnement multi-conteneurs

Solution : mettre en place un réseau docker avec la commande :

docker create network <nom du réseau> le réseau est nommé cirilgroup

J’ai essayé de mettre en place un fichier requirements.txt commun à tous les conteneurs, mais je n'ai pas réussi à résoudre cette erreur, malgré la modification du chemin du fichier dans les Dockerfiles des conteneurs et la reconstruction des images.