TP2 – NOSQL CASSANDRA RESTAURANT INSPECTIONS

Manon GARDIN

Matias OTTENSEN

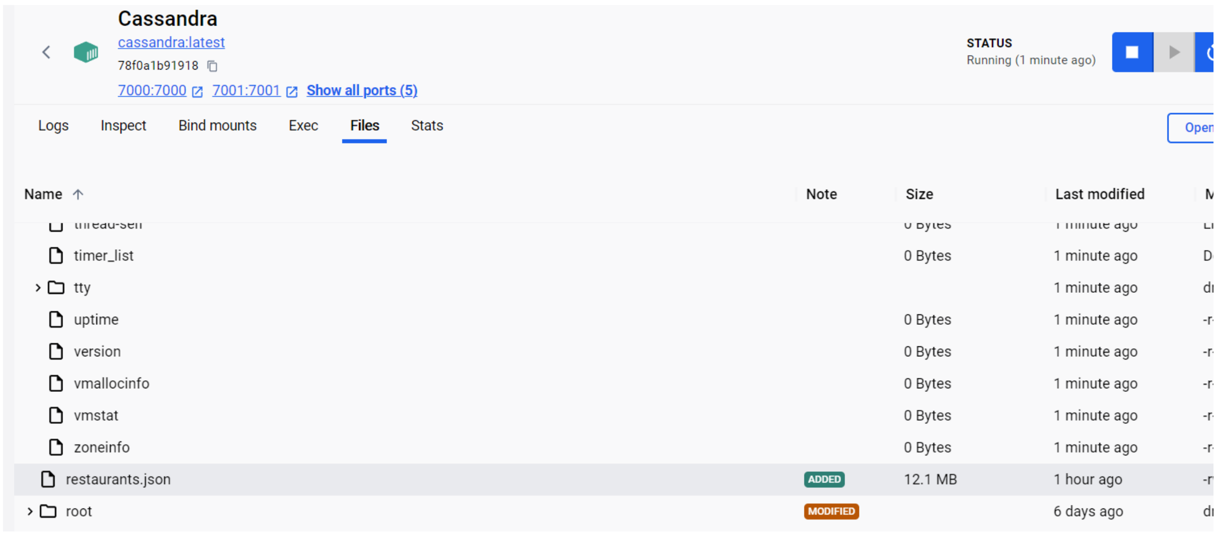
Alexandre GARNIER

Tiphaine KACHKACHI

Create the database

Files transfer

We drag and drop the restaurants.json into the files of our Cassandra container.



Cassandra
cassandra:latest
78f0a1b91918
7000:7000 7001:7001 Show all ports (5)
STATUS
Running (1 minute ago)

Name	Note	Size	Last modified	M
thread-self		0 Bytes	1 minute ago	L
timer_list		0 Bytes	1 minute ago	D
tty			1 minute ago	d
uptime		0 Bytes	1 minute ago	-r
version		0 Bytes	1 minute ago	-r
vmallocinfo		0 Bytes	1 minute ago	-r
vmstat		0 Bytes	1 minute ago	-r
zoneinfo		0 Bytes	1 minute ago	-r
restaurants.json	ADDED	12.1 MB	1 hour ago	-r
root	MODIFIED		6 days ago	d

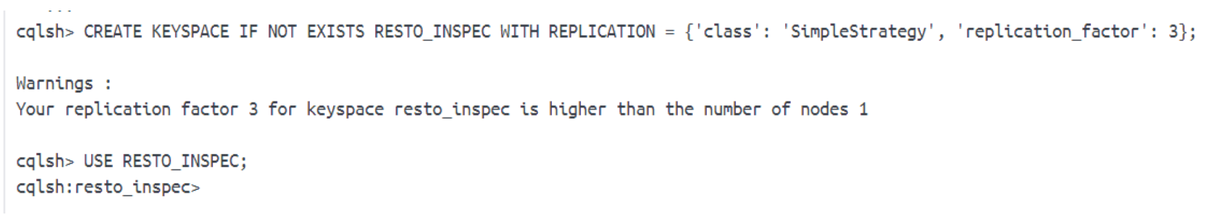
Create the keyspace

In the CLI, use the command :

```
CREATE KEYSPACE IF NOT EXISTS RESTO_INSPEC
WITH REPLICATION =
{ 'class': 'SimpleStrategy', 'replication_factor': 3 };
```

And then,

```
USE RESTO_INSPEC;
```



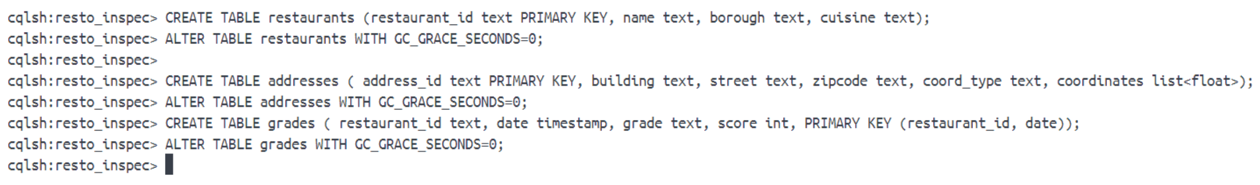
```
  ...
cqlsh> CREATE KEYSPACE IF NOT EXISTS RESTO_INSPEC WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 3};

Warnings :
Your replication factor 3 for keyspace resto_inspec is higher than the number of nodes 1

cqlsh> USE RESTO_INSPEC;
cqlsh:resto_inspec>
```

Create Tables

Let's generate the schema that mirrors the JSON structure provided.

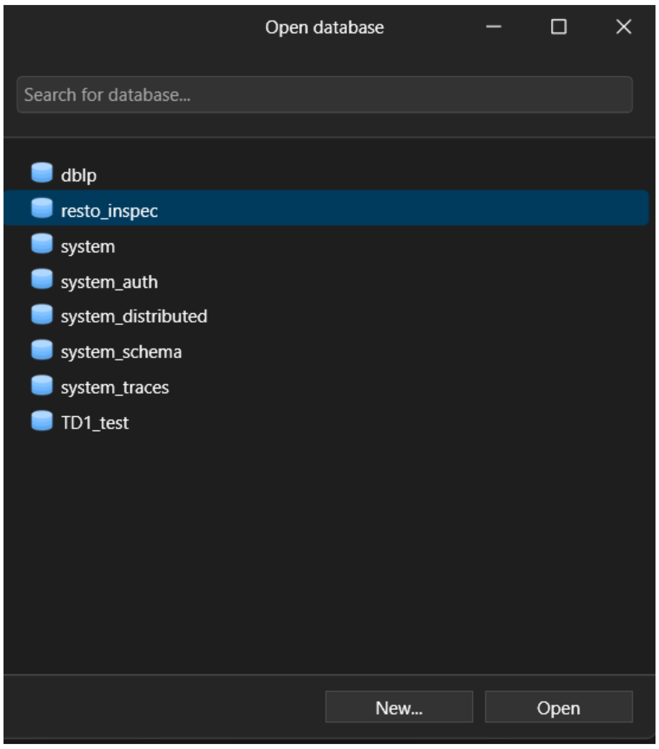
Create the tables in file CreaTable.sql :

```
CREATE TABLE restaurants (
      restaurant_id text PRIMARY KEY,
      name text,
      borough text,
      cuisine text
);
ALTER TABLE restaurants WITH GC_GRACE_SECONDS=0;


CREATE TABLE addresses (
    address_id text PRIMARY KEY,
    building text,
    street text,
    zipcode text,
    coord_type text,
    coord_X float,
    coord_Y float
);
ALTER TABLE addresses WITH GC_GRACE_SECONDS=0;

CREATE TABLE grades (
    restaurant_id text,
    date timestamp,
    grade text,
    score int,
    PRIMARY KEY (restaurant_id, date)
);
ALTER TABLE grades WITH GC_GRACE_SECONDS=0;
```



```
cqlsh:resto_inspec> CREATE TABLE restaurants (restaurant_id text PRIMARY KEY, name text, borough text, cuisine text);
cqlsh:resto_inspec> ALTER TABLE restaurants WITH GC_GRACE_SECONDS=0;
cqlsh:resto_inspec>
cqlsh:resto_inspec> CREATE TABLE addresses ( address_id text PRIMARY KEY, building text, street text, zipcode text, coord_type text, coordinates list<float>);
cqlsh:resto_inspec> ALTER TABLE addresses WITH GC_GRACE_SECONDS=0;
cqlsh:resto_inspec> CREATE TABLE grades ( restaurant_id text, date timestamp, grade text, score int, PRIMARY KEY (restaurant_id, date));
cqlsh:resto_inspec> ALTER TABLE grades WITH GC_GRACE_SECONDS=0;
cqlsh:resto_inspec> █
```

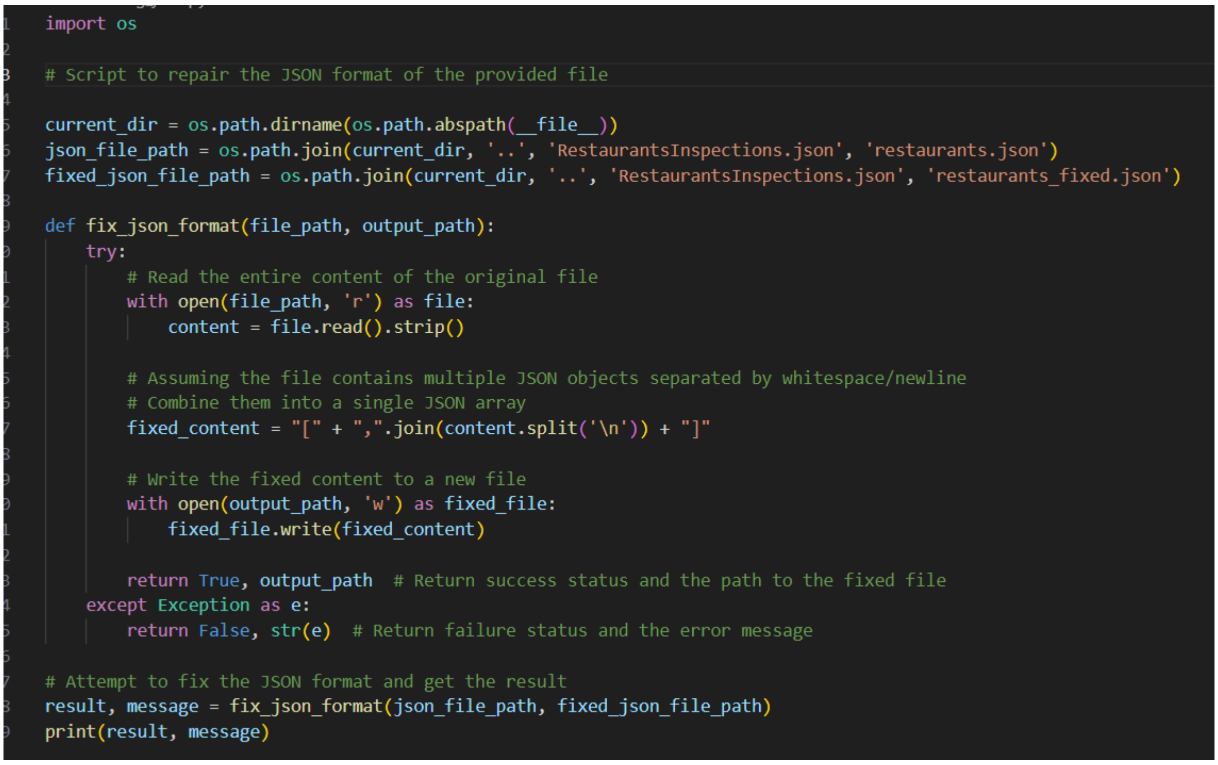
Now, we open TablePlus, and select the database we created.



Fixing Json file

We found out that the format of the Json is not correct, so we needed to do a script to correct the file.

We did the fixing_json.py :



```python
import os

# Script to repair the JSON format of the provided file

current_dir = os.path.dirname(os.path.abspath(__file__))
json_file_path = os.path.join(current_dir, '..', 'RestaurantsInspections.json', 'restaurants.json')
fixed_json_file_path = os.path.join(current_dir, '..', 'RestaurantsInspections.json', 'restaurants_fixed.json')

def fix_json_format(file_path, output_path):
    try:
        # Read the entire content of the original file
        with open(file_path, 'r') as file:
            content = file.read().strip()

        # Assuming the file contains multiple JSON objects separated by whitespace/newline
        # Combine them into a single JSON array
        fixed_content = "[" + ",".join(content.split('\n')) + "]"

        # Write the fixed content to a new file
        with open(output_path, 'w') as fixed_file:
            fixed_file.write(fixed_content)

        return True, output_path  # Return success status and the path to the fixed file
    except Exception as e:
        return False, str(e)  # Return failure status and the error message

# Attempt to fix the JSON format and get the result
result, message = fix_json_format(json_file_path, fixed_json_file_path)
print(result, message)
```

This script will create a new Json file, in order to not modify the original one.

Import the data

Then, we import the data_importation.py file in the Cassandra container.

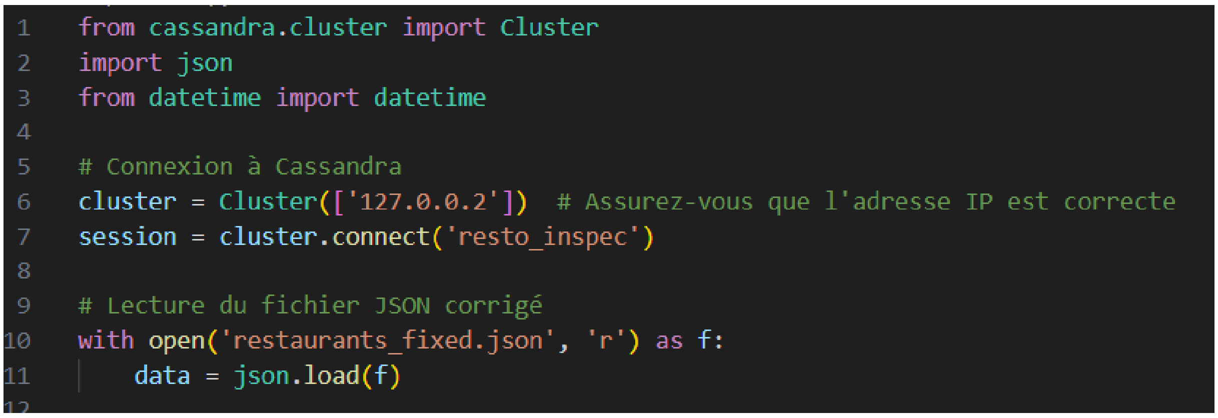
Before executing it, we will need to download python3 on the container, to have the good modules.

To do that :

```
docker exec -it Cassandra bash

apt-get update

apt-get install -y python3 python3-pip

pip3 install cassandra-driver
```

Now, we can execute the data_importation.py file in the Cassandra container.

In this code below, we setup the connection to the database :



```python
from cassandra.cluster import Cluster
import json
from datetime import datetime

# Connexion à Cassandra
cluster = Cluster(['127.0.0.2'])  # Assurez-vous que l'adresse IP est correcte
session = cluster.connect('resto_inspec')

# Lecture du fichier JSON corrigé
with open('restaurants_fixed.json', 'r') as f:
    data = json.load(f)
```

And then, we add the data with a query, for each table. (See in the data_importation.py file)

Here is the command :

docker exec -it Cassandra python3 data_importation.py