

**PROJET BIG DATA : CONSTRUCTION D'UN DATA LAKE AVEC HIVE**

**NOMBRE DES MEMBRES**

DIEKE ANGE JONATHAN

DOUDOU BI TOUVOLY ALAIN SERGE

SCHAEFER PHILIP

**ENCADRANTS**

MOPOLO GABRIEL

SERGIO SIMONIAN

## **SOMMAIRE**

### **OBJECTIF ET CONTEXTE DU PROJET**

#### **CHAPITRE 1 : CONSTRUCTION DU DATA LAKE**

##### **I- ARCHITECTURE BIG DATA**

- 1- Définition**
- 2- Présentation de notre architecture**
- 3- Chargement des données dans les différentes sources externes**

##### **II- MISE EN PLACE DU DATA LAKE AVEC HIVE**

- 1- Table interne**
- 2- Tables externes**

#### **CONCLUSION PARTIELLE**

#### **CHAPITRE 2 : TRAITEMENT DE DONNEE**

##### **I- MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL**

- 1- Installation de jupyter notebook dans la machine virtuelle**
- 2- Connexion à HIVE et récupération des données**

##### **II- CORRECTION DES DONNEES**

- 1- Traitement direct des données**
- 2- Traitement des données avec MapReduce**
- 3- La jointure entre catalogue et CO2**

#### **CONSLUSION PARTIELLE**

#### **CONCLUSION GENERALE**

## CONTEXTE ET OBJECTIFS DU PROJET

Contactés par un concessionnaire automobile, nous avons pour mission de l'aider à mieux cibler les véhicules susceptibles d'intéresser ses clients. Pour cela il met à notre disposition :

- son catalogue de véhicules (Catalogue.csv) ;
- son fichier clients contenant les achats de l'année en cours (Clients.csv - divisé en 2) ;
- un accès aux informations sur les immatriculations effectuées cette année (Immatriculations.csv) ;
- une brève documentation des données.

A la fin de ce projet nous devons proposer un outil permettant :

- d'évaluer en temps réel le type de véhicule le plus susceptible d'intéresser les clients qui se présenteront dans la concession ;
- d'envoyer une documentation précise sur le véhicule le plus adéquat pour des clients sélectionnés par son service marketing.

Pour la réalisation de ce projet, nous construisons notre data lake après avoir chargé les données dans les différentes sources (HDFS, MongoDB, Oracle NOSQL. Par la suite, nous traitons les données du fichier CO2.csv – ce fichier contient des informations sur certains véhicules du catalogue.

## Chapitre 1 : CONSTRUCTION DU DATA LAKE

### I- Architecture big data

#### 1. Définition

Une architecture Big Data est un ensemble de composants matériels et logiciels conçus pour traiter des quantités massives de données. Les architectures Big Data sont conçues pour être hautement évolutives et flexibles, et elles sont souvent mises en œuvre dans des environnements distribués pour permettre le traitement parallèle des données. Ces architectures peuvent stocker et traiter des données structurées, semi-structurées et non structurées, et sont conçues pour relever les défis de l'analyse de données à grande échelle.

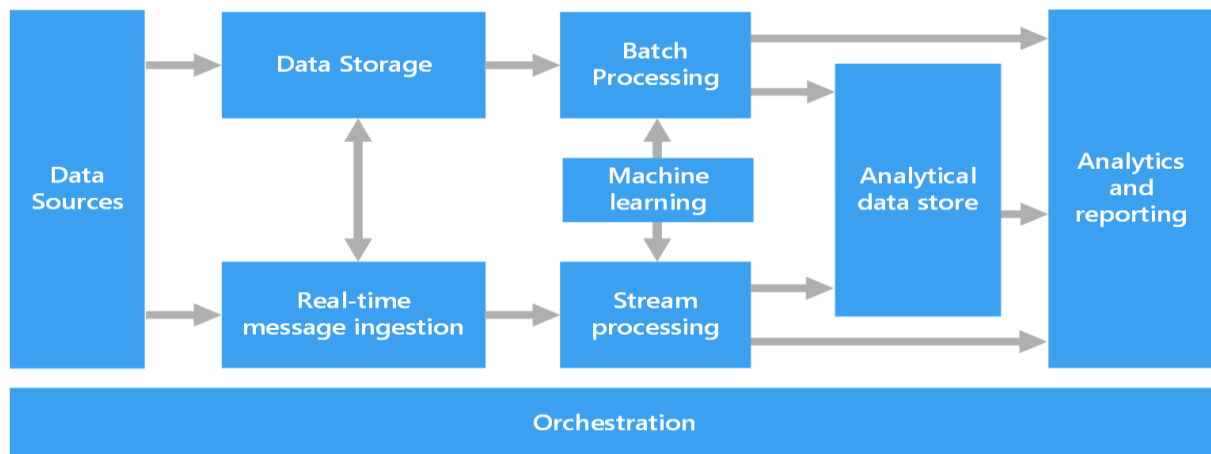


Figure 1 - Exemple d'architecture big data

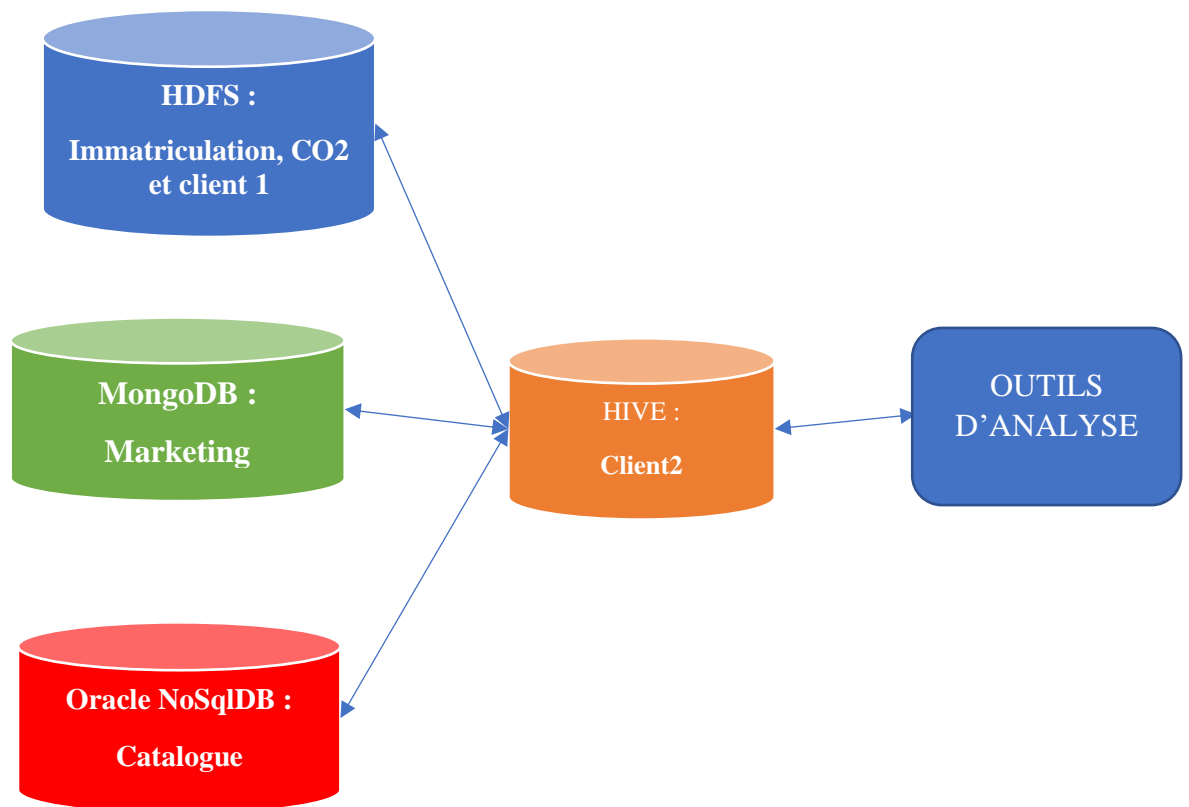
Source : [Microsoft Learn](#)

#### 2. Présentation de notre architecture

Notre architecture est celle d'un data lake mixte (physique et virtuelle) construite autour de HIVE qui fait office du frontal. Elle est composée des sources de données suivantes :

- ✓ HDFS – contient les fichiers Immatriculations.csv, CO2.csv et clients1.csv
- ✓ MongoDB – contient les données du fichier Marketing.csv
- ✓ Oracle NoSQL – contient les données de Catalogue.csv
- ✓ HIVE – contient les données de clients2.csv

A noter que Hive permet d'accéder aux données des autres sources. Ainsi, les tables ou fichiers en dehors de Hive sont des données virtuelles donc accessibles grâce à des tables externes et celles dans Hive sont des données physiques car physiquement présentes dans Hive.



*Figure 2 - Notre architecture data lake avec Hive*

### 3. Chargement des données dans les différentes sources externes

Nous présentons les scripts qui ont permis de créer et peupler les différentes bases de données et quelques résultats des requêtes de celles-ci.

L'environnement utilisé est celle d'une machine virtuelle Vagrant basée sur CentOS. Pour démarrer la machine, il suffit d'exécuter la commande suivante :

```
vagrant up
```

Et pour s'y connecter, exécuter la commande suivante :

```
vagrant ssh
```

La suite des commandes exécutées se fera dans la machine virtuelle

#### a. HDFS

On démarre d'abord **HDFS** en faisant :

```
Start-dfs.sh
```

Nous utilisons les commandes suivantes pour copier les fichiers (CO2.csv, Immatriculations.csv et clients.csv) dans **HDFS** :

```
hadoop fs -put /vagrant/VM_DATA/CO2 CO2
```

```
hadoop fs -put /vagrant/VM_DATA/Immatriculations Immatriculations
```

```
hadoop fs -put /vagrant/VM_DATA/client client1
```

L'option **-put** qui permet de spécifier que l'on souhaite copier un fichier ou un dossier local dans le système de fichiers distribué Hadoop prend deux arguments. Le premier argument est le chemin local vers le dossier ou le fichier que l'on souhaite copier et le second est le chemin Hadoop vers lequel on souhaite copier le dossier ou le fichier.

On vérifie que les fichiers sont bien présents

```
vagrant@oracle-21c-vagrant ~]$ hadoop fs -ls
Found 3 items
drwxr-xr-x - vagrant supergroup          0 2023-02-15 13:26 CO2
drwxr-xr-x - vagrant supergroup          0 2023-02-15 13:28 Client1
drwxr-xr-x - vagrant supergroup          0 2023-02-15 15:22 Immatriculations
vagrant@oracle-21c-vagrant ~]$
```

*Figure 3 - Résultats après le chargement des fichiers sur HDFS*

Afin de faciliter la suite des opérations, il a été fait le choix de placer chaque fichier dans un dossier dans lequel il est l'unique élément.

## **b. MongoDB**

Pour charger les données du fichier Marketing.csv dans MongoDB, nous utilisons l'utilitaire mongoimport :

```
mongoimport --db tpt --collection marketing --type csv --headerline --ignoreBlanks --file  
/vagrant/VM_DATA/Marketing/Marketing.csv
```

On se connecte à la base de données **tpt** et on vérifie si la collection **marketing** est bien présente

```

vagrant@oracle-21c-vagrant:~$ mongo
MongoDB shell version v3.4.24
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.24
server has startup warnings:
2023-03-10T13:31:32.548+0100 I CONTROL [initandlisten]
2023-03-10T13:31:32.548+0100 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2023-03-10T13:31:32.548+0100 I CONTROL [initandlisten] **      Read and write access to data and configuration is unrestricted.
2023-03-10T13:31:32.548+0100 I CONTROL [initandlisten]
> show dbs
admin 0.000GB
local 0.000GB
tpt 0.000GB
> use tpt
switched to db tpt
> show tables
marketing
> db.marketing.find()
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb69"), "age" : 21, "sexe" : "F", "taux" : 1396, "situationFamiliale" : "Celibataire", "nbEnfantsAcharge" : 0, "2eme
voiture" : "false" }
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb6a"), "age" : 35, "sexe" : "M", "taux" : 223, "situationFamiliale" : "Celibataire", "nbEnfantsAcharge" : 0, "2eme
voiture" : "false" }
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb6b"), "age" : 48, "sexe" : "M", "taux" : 401, "situationFamiliale" : "Celibataire", "nbEnfantsAcharge" : 0, "2eme
voiture" : "false" }
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb6c"), "age" : 26, "sexe" : "F", "taux" : 420, "situationFamiliale" : "En Couple", "nbEnfantsAcharge" : 3, "2eme vo
iture" : "true" }

```

Figure 4 - Résultats après le chargement des données dans MongoDB

### c. OracleNosql

Pour charger les données du Catalogue dans Oracle NOSQL, nous utilisons un programme java.

```

void loadCatalogueDataFromFile(String catalogueDataFileName) {
    InputStreamReader ipsr;
    BufferedReader br = null;
    InputStream ips;
    // Variables pour stocker les données lues d'un fichier.
    String ligne;
    System.out.println(x: "***** Dans : loadCatalogueDataFromFile *****");
    /* parcourir les lignes du fichier texte et découper chaque ligne */
    try {
        ips = new FileInputStream(catalogueDataFileName);
        ipsr = new InputStreamReader(ips);
        br = new BufferedReader(ipsr);
        while ((ligne = br.readLine()) != null) {
            if (ligne.contains(s: "marque") && ligne.contains(s: "nom") && ligne.contains(s: "prix") && ligne.contains(s: "couleur")) {
                continue;
            }
            ArrayList<String> catalogueRecord = new ArrayList<String>();
            StringTokenizer val = new StringTokenizer(ligne, delim: ",");
            while (val.hasMoreTokens()) {
                catalogueRecord.add(val.nextToken().toString());
            }
            String marque = catalogueRecord.get(index: 0);
            String nom = catalogueRecord.get(index: 1);
            String puissance = catalogueRecord.get(index: 2);
            String longueur = catalogueRecord.get(index: 3);
            String nbPlaces = catalogueRecord.get(index: 4);
            String nbPortes = catalogueRecord.get(index: 5);
            String couleur = catalogueRecord.get(index: 6);
            String occasion = catalogueRecord.get(index: 7);
            String prix = catalogueRecord.get(index: 8);
            // Add the catalogue in the KVStore
            this.insertCatalogueRow(marque, nom, Integer.parseInt(puissance), longueur, Integer.parseInt(nbPlaces), Integer.parseInt(nbPortes), couleur, Boolean.parseBoolean(
            ));
        }
        br.close();
        ipsr.close();
        ips.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Figure 5 - Extrait du programme (suppression et création de la table Catalogue)

## II- MISE EN PLACE DU DATA LAKE AVEC HIVE

Après avoir mis les données dans les différentes bases de données externes, on peut maintenant créer des tables externes à partir de Hive pour y avoir accès. Avant tout, nous démarrons HIVE et effectuons une connexion.

```
[vagrant@oracle-21c-vagrant ~]$ nohup hive --service metastore > /dev/null & nohup hiveserver2 > /dev/null &
[5] 10468
[6] 10469
[vagrant@oracle-21c-vagrant ~]$ nohup: ignoring input and redirecting stderr to stdout
nohup: ignoring input and redirecting stderr to stdout

[vagrant@oracle-21c-vagrant ~]$ beeline
Beeline version 2.3.9 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000
Connecting to jdbc:hive2://localhost:10000
Enter username for jdbc:hive2://localhost:10000:
Enter password for jdbc:hive2://localhost:10000:
Connected to: Apache Hive (version 3.1.3)
Driver: Hive JDBC (version 2.3.9)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000>
```

Figure 6 - Connexion à Hive

### 1- Table interne clients2

La requête de création de la table interne client2 dans Hive est la suivante :

```
CREATE TABLE client2 (
  age int,
  sexe string,
  taux int,
  situationFamiliale string,
  nbEnfantsAcharge int,
  2emeVoiture boolean,
  immatriculation string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED S TEXTFILE;
```

Figure 7 – création de la table interne client2 dans Hive

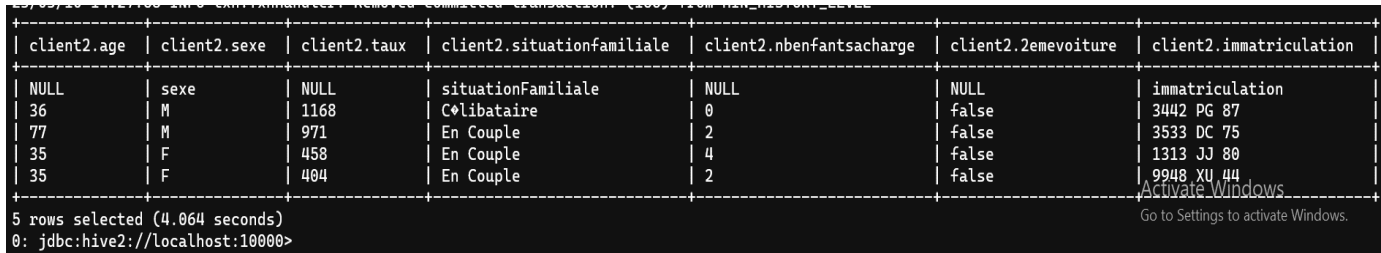


Une fois la table créée, il est possible de charger les données du fichier client2.csv via la commande suivante :

```
LOAD DATA LOCAL INPATH '/vagrant/TPT-data/Clients_62.csv' INTO TABLE clients2;
```

Vérifions que nous avons accès aux données :

```
select * from client2 limit 5;
```



client2.age	client2.sexe	client2.taux	client2.situationfamiliale	client2.nbenfantsacharge	client2.2emevoiture	client2.immatriculation
NULL	sexe	NULL	situationFamiliale	NULL	NULL	immatriculation
36	M	1168	Colibataire	0	false	3442 PG 87
77	M	971	En Couple	2	false	3533 DC 75
35	F	458	En Couple	4	false	1313 JJ 80
35	F	404	En Couple	2	false	9948 XU 44

Figure 8 – affichage des 5 premières lignes de clients1

## 2- Tables externes

### ✓ Table externe Marketing vers MongoDB

La requête de création de la table externe pointant vers la collection marketing dans MongoDB est la suivante :

```
drop table marketing_hive_mongo_ext ;
CREATE EXTERNAL TABLE marketing_hive_mongo_ext (
  id int,
  age int,
  sexe string,
  taux int,
  situationFamiliale string,
  nbEnfantsAcharge int,
  2emeVoiture string
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
WITH SERDEPROPERTIES('mongo.columns.mapping'='{ "id": "_id", "age": "age",
"sexe": "sexe", "taux": "taux", "situationFamiliale": "situationFamiliale",
"nbEnfantsAcharge": "nbEnfantsAcharge", "2emeVoiture": "2eme voiture"}')
TBLPROPERTIES ('mongo.uri'='mongodb://localhost:27017/tpt.marketing');
```

Figure 9 - Création de la table externe marketing

Vérifions que nous avons accès aux données :

```
select * from marketing_hive_mongo_ext limit 5;
```

```

0: jdbc:hive2://localhost:10000>
0: jdbc:hive2://localhost:10000> select * from marketing_hive_mongo_ext limit 5;
+-----+-----+-----+-----+-----+
| marketing_hive_mongo_ext.id | marketing_hive_mongo_ext.age | marketing_hive_mongo_ext.sexe | marketing_hive_mongo_ext.taux | marketing_hive_mongo_ext.situationfamiliale | marketing_hive_mongo_ext.nbenfantsacharge | marketing_hive_mongo_ext.2emevoiture | marketing_hive_mongo_ext. |
+-----+-----+-----+-----+-----+
| NULL | 0 | 21 | F | false | 1396 | | C°libataire |
| NULL | 0 | 35 | M | false | 223 | | C°libataire |
| NULL | 0 | 48 | M | false | 401 | | C°libataire |
| NULL | 3 | 26 | F | true | 420 | | En Couple |
| NULL | 2 | 27 | F | false | 153 | | En Couple |
+-----+-----+-----+-----+-----+
5 rows selected (8.546 seconds)
0: jdbc:hive2://localhost:10000>

```

Figure 10 - affichage des 5 premières lignes de la table externe marketing

## ✓ Tables externes vers HDFS

### ➤ Client1

La requête de création de la table externe pointant vers le fichier clients1.csv dans HDFS est la suivante :

```

drop table clients1_hive_ext;

CREATE EXTERNAL TABLE clients1_hive_ext(
  age int,
  sexe string,
  taux int,
  situationFamiliale string,
  nbEnfantsAcharge int,
  2emeVoiture boolean,
  immatriculation string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs:/Client1'
TBLPROPERTIES ("skip.header.line.count"="1");

```

Figure 11 - Création de la table externe clients1

Vérifions que nous avons accès aux données :

```
select * from clients1_hive_ext limit 5;
```

clients1_hive_ext.age	clients1_hive_ext.sexe	clients1_hive_ext.taux	clients1_hive_ext.situationfamiliale	clients1_hive_ext.nbenfantsacharge
62	M	6290 DM 24	1262	En Couple
false	M	7530 VH 52	514	En Couple
68	M	7168 HX 32	181	En Couple
false	F	1539 UR 49	829	En Couple
26	M	4738 YG 76	1169	En Couple
true	M			En Couple
34	M			En Couple
false	M			En Couple
50	M			En Couple
false	M			En Couple

5 rows selected (1.365 seconds)  
0: jdbc:hive2://localhost:10000>

Figure 12 - affichage des 5 premières lignes de la table externe clients1

### ➤ Immatriculations

La requête de création de la table externe pointant vers le fichier immatriculations.csv dans HDFS est la suivante :

```
drop table immatriculations_hive_ext;
CREATE EXTERNAL TABLE immatriculations_hive_ext(
  immatriculation string,
  marque string,
  nom string,
  puissance int,
  longueur string,
  nbPlaces int,
  nbPortes int,
  couleur string,
  occasion boolean,
  prix int
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs:/Immatriculations'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Figure 13 - Création de la table externe immatriculations

Vérifions que nous avons accès aux données :

```
select * from immatriculations_hive_ext limit 5;
```

immatriculations_hive_ext.immatriculation	immatriculations_hive_ext.marque	immatriculations_hive_ext.nom	immatriculations_hive_ext.puissance	immatriculations_hive_ext.longueur	immatriculations_hive_ext.nbplaces	immatriculations_hive_ext.nbportes	immatriculations_hive_ext.couleur	immatriculations_hive_ext.occasion	immatriculations_hive_ext.prix
3176 TS 67	Renault	Laguna 2.0T	170	27300	5	5	blanc	false	1
3721 QS 49	Volvo	S80 T6	272	50500	5	5	noir	false	t
9099 UV 26	Volkswagen	Golf 2.0 FSI	150	16029	5	5	gris	true	m
3563 LA 55	Peugeot	1007 1.4	75	9625	5	5	blanc	true	c
6963 AX 34	Audi	A2 1.4	75	18310	5	5	gris	false	c

5 rows selected (0.513 seconds)  
0: jdbc:hive2://localhost:10000>

Figure 14 - affichage des 5 premières lignes de la table externe immatriculations

### ➤ Table externe CO2

La requête de création de la table externe pointant vers le fichier immatriculations.csv dans HDFS est la suivante :

```
drop table CO2_hive_ext;

CREATE EXTERNAL TABLE CO2_hive_ext(
  id int,
  marque_modele string,
  bonus_malus string,
  rejetsCO2gkm string,
  coutenergie string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs:/CO2'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Figure 15 - Création de la table externe CO2

Vérifions que nous avons accès aux données :

```
select * from CO2_hive_ext limit 5;
```

co2_hive_ext.id	co2_hive_ext.marque_modele	co2_hive_ext.bonus_malus	co2_hive_ext.rejetsco2gkm	co2_hive_ext.coutenergie
2	AUDI E-TRON SPORTBACK 55 (408ch) quattro	-6Å 000Å, -Å 1	0	319Å Å, -
3	AUDI E-TRON SPORTBACK 50 (313ch) quattro	-6Å 000Å, -Å 1	0	356Å Å, -
4	AUDI E-TRON 55 (408ch) quattro	-6Å 000Å, -Å 1	0	357Å Å, -
5	AUDI E-TRON 50 (313ch) quattro	-6Å 000Å, -Å 1	0	356Å Å, -
6	BMW i3 120 Ah	-6Å 000Å, -Å 1	0	204Å Å, -

5 rows selected (0.381 seconds)  
0: jdbc:hive2://localhost:10000>

Figure 16 - affichage des 5 premières lignes de la table externe CO2

### ✓ Table externe Catalogue vers Oracle NOSQL

La requête de création de la table externe pointant vers la table catalogue dans Oracle NOSQL est la suivante :

```
drop table catalogue_hive_ext;
CREATE EXTERNAL TABLE catalogue_hive_ext (
    marque string,
    nom string,
    puissance int,
    longueur string,
    nbPlaces int,
    nbPortes int,
    couleur string,
    occasion boolean,
    prix int,
    id int)
STORED BY 'oracle.kv.hadoop.hive.table.TableStorageHandler'
TBLPROPERTIES (
    "oracle.kv.kvstore" = "kvstore",
    "oracle.kv.hosts" = "localhost:5000",
    "oracle.kv.hadoop.hosts" = "localhost/127.0.0.1",
    "oracle.kv.tableName" = "catalogue");
```

Figure 17 - Création de la table externe catalogue

Vérifions que nous avons accès aux données :

```
select * from catalogue_hive_ext limit 5;
```

catalogue_hive_ext.marque	catalogue_hive_ext.nom	catalogue_hive_ext.puissance	catalogue_hive_ext.longueur	catalogue_hive_ext.nbplaces	catalogue_hive_ext.nbportes	catalogue_hive_ext.couleur	catalogue_hive_ext.occasion	catalogue_hive_ext.prix	catalogue_hive_ext.id
Volkswagen	New Beetle 1.8	110	moyenne	5		gris	false	26630	3.7E-37
Volkswagen	Golf 2.0 FSI	150	moyenne	5		blanc	true	16029	4.4E-37
Seat	Toledo 1.6	102	longue	5		gris	false	18880	5.9E-37
Seat	Toledo 1.6	102	longue	5		bleu	false	18880	6.0E-37
Saab	9.3 1.8T	150	longue	5		gris	true	27020	6.3E-37

rows selected (2.153 seconds)  
: jdbc:hive2://localhost:10000>

Activate Windows  
Go to Settings to activate Windows.

Figure 18 - affichage des 5 premières lignes de la table externe catalogue

## Conclusion partielle

Dans cette partie, nous avons défini l'architecture de notre dataLake et effectué sa mise en place. Nous avons chargé les différents fichiers dans les différentes sources de données comme défini tantôt. Ensuite nous avons créé les tables externes dans Hive pour y avoir accès. De plus, nous avons effectué des requêtes de consultation pour nous s'assurer que les données étaient bien accessibles.

Dans la suite, nous allons utiliser un Jupyter Notebook pour traiter les données de CO2.

## Chapitre 2 : TRAITEMENT DES DONNEES

Dans cette section nous nous montrons comment a été faite la connexion au data lake grâce à Python de même que les manipulations effectuées pour le traitement des données de la table CO2.

### I. Mise en place de l'environnement de travail

#### 1. Installation de Jupyter Notebook dans la machine virtuelle

Commençons par installer les dépendances de *pip* puis installons Jupyter :

```
-- Installation des dépendances pour pip  
curl https://bootstrap.pypa.io/pip/3.6/get-pip.py -o get-pip.py  
python3 get-pip.py --user  
  
-- Installation de Jupyter  
~/local/bin/pip install jupyter
```

Ceci étant fait, il a aussi fallu ajouté une ligne de configuration dans le Vagrantfile de la machine virtuelle afin de faire la redirection de port entre la machine hôte et la VM. Il est nécessaire de redémarrer la machine afin que les modifications apportées soient prises en compte.

```
Config.vm.network "forwarded\_port", guest: 8888, host: 8888
```

Après l'installation, on lance jupyter Notebook avec la commande

```
jupyter notebook --ip=0.0.0.0
```

```
D:\TPMONGO\TP_VM\vagrant-projects\OracleDatabase\21.3.0>vagrant ssh

Welcome to Oracle Linux Server release 8.7 (GNU/Linux 5.15.0-5.76.5.1.el8uek.x86_64)

The Oracle Linux End-User License Agreement can be viewed here:

 * /usr/share/eula/eula.en_US

For additional packages, updates, documentation and community help, see:

 * https://yum.oracle.com/

Last login: Fri Mar 10 15:49:11 2023 from 10.0.2.2
[vagrant@oracle-21c-vagrant ~]$ jupyter notebook --ip=0.0.0.0
[I 16:18:03.453 NotebookApp] Serving notebooks from local directory: /home/vagrant
[I 16:18:03.453 NotebookApp] Jupyter Notebook 6.4.10 is running at:
[I 16:18:03.453 NotebookApp] http://oracle-21c-vagrant:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee
[I 16:18:03.453 NotebookApp] or http://127.0.0.1:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee
[I 16:18:03.453 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to :
[W 16:18:03.464 NotebookApp] No web browser found: could not locate runnable browser.
[C 16:18:03.465 NotebookApp]

To access the notebook, open this file in a browser:
    file:///home/vagrant/.local/share/jupyter/runtime/nbserver-13208-open.html
Or copy and paste one of these URLs:
    http://oracle-21c-vagrant:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee
    or http://127.0.0.1:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee
```

Figure 19 - Lancement de Jupyter

Nous obtenons une url locale qui nous permet d'ouvrir jupyter notebook :

<http://127.0.0.1:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee>

## 2. Connexion à Hive et récupération des données

Notre objectif dans ce notebook est de traiter les données de CO2 et effectuer leur intégration dans la table catalogue. Pour ce faire, nous effectuons une connexion à partir de Hive pour charger les données de CO2 et les manipuler.

### ✓ Pyspark

PySpark est une bibliothèque Spark écrite en Python pour exécuter des applications Python en utilisant les capacités d'Apache Spark. PySpark permet d'exécuter des applications en parallèle sur un cluster distribué (plusieurs nœuds). En d'autres termes, PySpark est une API Python pour Apache Spark. Apache Spark est un moteur de traitement analytique pour le traitement de données distribuées puissantes à grande échelle et les applications d'apprentissage automatique.

Source : [Spark by examples](#)

### ✓ PyHive

Pour la connexion à Hive à partir de Python, nous utilisons **pyhive**. C'est est une collection d'interfaces Python DB-API et SQLAlchemy pour Presto et Hive. Il permet d'interroger des bases de données à partir de Python.

```
from pyhive import hive
conn = hive.connect(host='localhost', port=10000)
cursor = conn.cursor()
```



Nous importons le module hive de pyhive et effectuons une connexion vers notre machine locale.

```

from pyhive import hive
conn = hive.connect(host='localhost', port=10000)
cursor = conn.cursor()

def import_data(table_name):
    cursor.execute(f"SELECT * FROM {table_name}")
    result = cursor.fetchall()
    data = pd.DataFrame(result, columns=[desc[0] for desc in cursor.description])
    # Column transformation
    col = data.columns
    col = col.str.replace(f'{table_name}.', '')
    data.columns = col
    data = spark.createDataFrame(data)

    return data

```

Nous définissons ensuite une fonction pour récupérer les données distances à partir de HIVE. La fonction prend le nom d'une table comme argument, effectue une sélection pour récupérer les données et transforme le résultat en dataframe de Spark.

On utilise cette fonction pour récupérer les données de **CO2** en passant en paramètre le nom de la table externe.

```
co2 = import_data('co2_hive_ext')
```

```
co2.show()
```

Ensuite on affiche le résultat :

id	marque_modele	bonus_malus	rejetsco2gkm	coutenergie
2	AUDI E-TRON SPORT...	-6Â 000â, -Â 1	0	319Â â, -
3	AUDI E-TRON SPORT...	-6Â 000â, -Â 1	0	356Â â, -
4	AUDI E-TRON 55 (4...	-6Â 000â, -Â 1	0	357Â â, -
5	AUDI E-TRON 50 (3...	-6Â 000â, -Â 1	0	356Â â, -
6	BMW i3 120 Ah	-6Â 000â, -Â 1	0	204Â â, -
7	BMW i3s 120 Ah	-6Â 000â, -Â 1	0	204Â â, -
8	CITROEN BERLINGO	-6Â 000â, -Â 1	0	203Â â, -
9	CITROEN C-ZERO	-6Â 000â, -Â 1	0	491Â â, -
10	DS DS3 CROSSBACK ...	-6Â 000â, -Â 1	0	251Â â, -
11	HYUNDAI KONA elec...	-6Â 000â, -Â 1	0	205Â â, -
12	HYUNDAI KONA elec...	-6Â 000â, -Â 1	0	205Â â, -
13	JAGUAR I-PACE EV4...	-6Â 000â, -Â 1	0	271Â â, -
14	"KIA e-NIRO Moteu...	150kW (204ch)"	-6Â 000â, -Â 1	0
15	"KIA e-NIRO Moteu...	100kW (136ch)"	-6Â 000â, -Â 1	0
16	KIA SOUL Moteur Â...	-6Â 000â, -Â 1	0	214Â â, -
17	KIA SOUL Moteur Â...	-6Â 000â, -Â 1	0	214Â â, -
18	MERCEDES EQC 400 ...	-6Â 000â, -Â 1	0	291Â â, -
19	MERCEDES VITO Tou...	-6Â 000â, -Â 1	0	411Â â, -
20	MERCEDES VITO Tou...	-6Â 000â, -Â 1	0	411Â â, -
21	MINI MINI Cooper ...	-6Â 000â, -Â 1	0	199Â â, -

On remarque que la *marque* et le *modèle* sont dans la même colonne, les valeurs de *bonus-malus* et de *cout énergie* ont des caractères qui ne sont pas interprétables.

## II. CORRECTION DES DONNEES

Pour le traitement des données nous avons envisager deux approches. La première consiste à utiliser pyspark pour manipuler les données directement et la deuxième consiste à utiliser un programme map/reduce.

### 1. Traitement direct des données

#### a. Marque model

Nous commençons par diviser la colonne `marque_model` en deux.

```
: ► split_col = split(co2['marque_model'], ' ')\n\n: ► marque = split_col.getItem(0)\n\n: ► co2 = co2.withColumn('marque', marque)
```

La marque étant les premières valeurs dans la colonne `marque_model`, on effectue un split suivant l'espace et on récupère la première valeur d'indice 0. Le reste constitue le modèle. Nous obtenons ainsi deux colonnes *marque* et *model*.

On observe les valeurs distinctes de marques ceux de modèle

```
co2.select("marque").distinct().show()
```

marque
MERCEDES
PORSCHE
HYUNDAI
TOYOTA
SKODA
NISSAN
LAND
CITROEN
BENTLEY
AUDI
MINI
PEUGEOT
JAGUAR
VOLVO
TESLA
BMW
VOLKSWAGEN
KIA
SMART
RENAULT

only showing top 20 rows

```
co2.select("model").show()
```

model
E-TRON SPORTBACK ...
E-TRON SPORTBACK ...
E-TRON 55 (408ch)...
E-TRON 50 (313ch)...
i3 120 Ah
i3s 120 Ah
BERLINGO
C-ZERO
DS3 CROSSBACK E-T...
KONA electric 64 kWh
KONA electric 39 kWh
I-PACE EV400 (400...
SOUL Moteur ÃfÃ0l...
SOUL Moteur ÃfÃ0l...
EQC 400 4MATIC
VITO Tourer long ...
VITO Tourer extra...
MINI Cooper SE Ha...
Nouvelle NISSAN LEAF
Nouvelle NISSAN LEAF

only showing top 20 rows

Dans les deux cas, seulement les 20 premières lignes sont affichées.

## b. bonus-malus

```
❏ co2.select('bonus_malus').distinct().show()
```

```
+-----+
| bonus_malus |
+-----+
| +7Â 613â,- |
| -6Â 000â,-Â 1 |
| +7Â 890â,- |
| +7Â 340â,- |
| +6Â 810â,- |
| - |
| +7Â 073â,- |
| +8Â 460â,- |
| +8Â 173â,- |
| +8Â 753â,- |
+-----+
```

Nous remarquons que les données de bonus-malus ont des caractères spéciaux qui ne sont pas interprétable. Au lieu de ça « -6Â 000â,-Â 1 » nous devons plutôt avoir « -6000 € 1 », de même pour les autres. Cependant comme nous travaillons avec des données numériques, nous allons avoir **-6000**. On fera pareil pour les autres valeurs. De plus, nous avons des valeurs manquantes représenter par le symbole « - ». Nous remplaçons les valeurs manquantes par la valeur de bonus-malus la plus fréquente (**le mode**).

Ainsi après traitement des données erronées nous avons obtenus :

```
❏ co2.select('bonus_malus').distinct().show()
```

```
+-----+
| bonus_malus |
+-----+
| 7340 |
| 7613 |
| -6000 |
| 6810 |
| 7890 |
| 8753 |
| 7073 |
| 8460 |
| 8173 |
+-----+
```

Les valeurs de bonus-malus ont été traitées maintenant propres pour l'usage. ;  
Nous faisons le traitement sur la colonne « cout énergie »

Avant traitement	Après traitement
<pre>co2.select('coutenergie').distinct</pre> <pre>+-----+</pre> <pre> coutenergie </pre> <pre>+-----+</pre> <pre>67Â â,- </pre> <pre>247Â â,- </pre> <pre>66Â â,- </pre> <pre>85Â â,- </pre> <pre>47Â â,- </pre> <pre>43Â â,- </pre> <pre>221Â â,- </pre> <pre>204Â â,- </pre> <pre>214Â â,- </pre> <pre>74Â â,- </pre> <pre>185Â â,- </pre> <pre>199Â â,- </pre> <pre>77Â â,- </pre> <pre>491Â â,- </pre> <pre>78Â â,- </pre> <pre>206Â â,- </pre> <pre>725Â â,- </pre> <pre>271Â â,- </pre> <pre>177Â â,- </pre> <pre>357Â â,- </pre> <pre>+-----+</pre> <pre>only showing top 20 rows</pre>	<pre>co2.select('coutenergie').distinct</pre> <pre>+-----+</pre> <pre> coutenergie </pre> <pre>+-----+</pre> <pre>251 </pre> <pre>85 </pre> <pre>53 </pre> <pre>78 </pre> <pre>155 </pre> <pre>210 </pre> <pre>271 </pre> <pre>209 </pre> <pre>319 </pre> <pre>725 </pre> <pre>47 </pre> <pre>177 </pre> <pre>185 </pre> <pre>291 </pre> <pre>206 </pre> <pre>715 </pre> <pre>205 </pre> <pre>54 </pre> <pre>491 </pre> <pre>235 </pre> <pre>+-----+</pre> <pre>only showing top 20 rows</pre>

### III. Traitement des données avec un programme MapReduce

Map/reduce est un modèle de programmation utilisé pour traiter de grandes quantités de données parallèlement sur des clusters de machines. Il divise le travail en deux étapes principales : mapper une fonction pour chaque élément de données, produisant une liste de paires clé-valeur, puis réduire les paires clé-valeur en une liste de résultats. Les opérations sont effectuées en parallèle sur différents nœuds du cluster, permettant de traiter rapidement de grandes quantités de données. MapReduce est utilisé dans des systèmes de traitement de données distribués tels que Hadoop et Apache Spark pour l'analyse de données, le traitement de logs, la recherche web, etc.

Source : [Apache Map/Reduce](#)

Dans notre cas, nous effectuons une fonction mapper.

```

def mapper(line):
    line = tuple(line)

    # colonne marque
    marque = line[1].split(" ")[0].replace("\"", "").capitalize() #replace quote near Volkswagen
    # colonne Malus/Bonus
    if line[2] in ["150kW (204ch)", "100kW (136ch)"]:
        return (None, None)
    #
    malus_bonus = line[2].strip().replace("Â\xa0", "").replace(r"[\d-]+", "").replace("€", "").replace("\"", "")
    malus_bonus = line[2].strip().replace("Â\xa0", "").replace(r"[\d-]+", "").replace("€", "").replace("\"", "")
    malus_bonus = ''.join(re.findall(r"[\d-]+", malus_bonus))

    # missing values bonus_malus
    malus_bonus = "0" if len(malus_bonus) == 1 else malus_bonus
    # colonne cout energie
    cout = line[-1]
    coutSplitted = cout.replace('Â', '').split()
    cout = coutSplitted[0] + coutSplitted[1] if len(coutSplitted) == 3 else coutSplitted[0] if len(coutSplitted) == 2 else
    cout = cout.encode('ascii', 'ignore').decode('ascii').replace(',', '')

    # colonne Rejet CO2
    rejet = line[3]

    malus_bonusInt = int(malus_bonus)
    rejetInt = (rejet)
    coutInt = int(cout)

    # on crée le couple key value avec la marque comme key
    new_value = f"{malus_bonusInt}|{rejetInt}|{coutInt}"
    return (marque, new_value)

```

Cette fonction *mapper* prend en entrée une ligne de données et applique un ensemble d'opérations pour extraire les informations nécessaires. Elle renvoie un couple (key, value) où la clé (key) est la marque de la voiture extraite à partir de la deuxième colonne, et la valeur (value) est une chaîne de caractères contenant le coût de l'énergie, le rejet de CO2, et le bonus/malus pour la voiture extraits des colonnes correspondantes. Si la colonne Malus/Bonus contient des valeurs spécifiques, la fonction renvoie None, None pour ignorer ces lignes MapReduce.

Après la phase de map nous devons effectuer une fonction reducer qui va mettre les pairs de clé ensemble pour reconstituer notre dataset de CO2. Cependant, dans notre approche, nous nous retrouvons avec une fonction map qui nous retourne le résultat que l'on cherche. Du coup, de façon exceptionnelle, nous n'allons pas effectuer de fonction reducer.

#### IV. Jointure entre catalogue et CO2

Pour effectuer la jointure, nous importons le fichier catalogue à l'aide de la fonction *import\_data* créer précédemment.

```

catalogue = catalogue.join(co2, "marque", "left")

```

Nous avons fait un left join en gardant catalogue comme base. La jointure s'est faite sur la marque de véhicule.

Entrée [49]: `catalogue.toPandas()`

	marque	nom	puissance	longueur	nbplaces	nbportes	couleur	occasion	prix	id	id	bonus_malus	rejetsco2gkm	coutenergie	model
0	MERCEDES	S500	306	très longue	5	5	rouge	False	101300	0E-18	459.0	8753.0	262	1051.0	SPRINTER Combi 319 CDI (190ch) 4x4 PTAC 3500 k...
1	MERCEDES	S500	306	très longue	5	5	rouge	False	101300	0E-18	458.0	8753.0	262	1051.0	SPRINTER Combi 319 CDI (190ch) 4x4 PTAC 3500 k...
2	MERCEDES	S500	306	très longue	5	5	rouge	False	101300	0E-18	457.0	8753.0	260	1041.0	SPRINTER Combi 319 CDI (190ch) 4x4 PTAC 3500 k...

Entrée [ ]:

Pour bien visualiser les données, nous avons utilisé le dataframe de pandas.

### Conclusion partielle

Nous avons dans cette partie effectué une connexion à HIVE à partir un notebook Jupyter. Ensuite, nous avons importé les données et effectué le traitement suivant deux approches qui mènent au même résultat. Enfin, nous avons effectué une jointure entre les données de CO2 et catalogue.

## **Conclusion générale**

Nous avons dans un premier temps mis en place une architecture big data pour la création de notre data lake. Pour ce faire, nous avons chargé les données dans les différentes sources. Puis Hive a été utilisé comme frontal pour créer des tables internes et externes pour manipuler les données physiques et virtuelles. Dans un second temps, une connection a été établie avec le data lake à partir d'un notebook Python, dans lequel, nous avons utilisé pyhive pour la connexion à hive et pyspark pour la manipulation et le traitement des données. De plus, nous avons joint les données de CO2 et de catalogue. Cette architecture, nous a permis de manipuler des données provenant de diverses sources à partir d'un seul frontal. Ce data lake sera utilisé par la suite pour développer les algorithmes de machine learning, faire les analyses et des prédictions