

**Projet de Modélisation de Données - Entrepôt de Données avec PySpark, Hive, Power Bi****Description du Projet**

Ce projet vise à nettoyer, transformer et modéliser des données brutes provenant de différentes sources (clients, vélos, magasins de vélos et commandes) pour créer un entrepôt de données (Data Warehouse). Les données nettoyées et modélisées sont ensuite stockées dans HDFS en format Parquet et exploitées via Hive et Power BI pour des analyses de type Business Intelligence (BI).

**Structure du Projet**

```
├─ data/ # Dossier pour les fichiers de données brutes (CSV ou autres formats).
├─ scripts/ # Contient les scripts PySpark pour le nettoyage et la transformation des données.
|   ├── Clean_customers2.py # Script de nettoyage des données clients.
|   ├── clean_orders2.py # Script de nettoyage des données des commandes.
|   ├── clean_bikes.py # Script de nettoyage des données des vélos.
|   ├── clean_bikeshops.py # Script de nettoyage des données des vélos.
|   ├── create_star_schema2.py # Script pour créer la table des faits avec jointures.
|   └── create_calendar_lookup2.py # Script pour créer la table de lookup calendrier.
├─ hive/ # Scripts Hive pour la création des tables et leur accès depuis Hive/Impala.
|   ├── Creating_Hive_tables # Script utilisées dans Hue pour créer les tables Hive externes
|   └── Project_hive_queries # Scripts Test pour requêter les tables dans Hive/Impala
└─ JAR/ # Contenant les fichiers JAR pour dépendances Spark nécessaires au projet
```

**Environnement :**

- **Cloudera Quickstart VM** (VirtualBox ou VMware)
- **Spark 1.6**
- **Hadoop 2.6**
- **Hive 1.1.0**
- **Python 2.6.6**
- **Impala** pour les requêtes BI et connecteur Power BI

## Instructions d'Exécution

### Organisation de la structure du projet dans Cloudera et HDF :

Pour ce projet, j'ai opté de créer 3 répertoires dans HDFS :

- user/cloudera/Raw\_data/Project
- user/cloudera/Clean\_data/Project
- user/cloudera/Modeled\_data/Project

Pour les fichiers parquets, ils seront sauvegardés dans leurs propre répertoire également :

- user/cloudera/clean\_data/Project/bikes\_parquet
- user/cloudera/clean\_data/Project/bikeshops\_parquet
- user/cloudera/clean\_data/Project/orders\_parquet
- user/cloudera/clean\_data/Project/Customers\_parquet

Après le nettoyage et la création de la table fact\_orders et dimensions liées, les fichiers parquets modelées seront sauvegardées dans les repertoires :

- user/cloudera/modeled\_data/Project/bikes\_lookup.parquet
- user/cloudera/modeled\_data/Project/bikeshops\_lookup.parquet
- user/cloudera/modeled\_data/Project/orders\_lookup.parquet
- user/cloudera/modeled\_data/Project/customers\_lookup.parquet
- user/cloudera/modeled\_data/Project/calendar\_lookup.parquet

les scripts seront chargées depuis la machine local vers la virtual box tel que :

- C:\Users\khali\PMN\Project\_Final\Data\ étant le dossier sur ma machine locale contenant les fichiers d'origine.
- /home/cloudera/Project étant le dossier crée sur cloudera pour rassembler les fichiers csv avant de les importer dans HDFS, et ainsi qu'emplacement pour les scripts py.

### Étape 1 : Chargement des Données Brutes

Placer les fichiers CSV bruts dans HDFS. Dans ce projet, j'ai paramétré un port pour pouvoir faire le transfert de tout fichiers depuis ma machine à la machine virtuelle de VirtualBox.

Ayant fait la configuration du port, il faut entrer la commande suivante dans un terminal Windows :

```
scp -P 2222 "C:\Users\khali\PMN\Project_Final\Data\Customers.csv"  
cloudera@localhost:/home/cloudera/Project/
```

```
scp -P 2222 "C:\Users\khali\PMN\Project_Final\Data\.bikes.csv"  
cloudera@localhost:/home/cloudera/Project/
```

```
scp -P 2222 "C:\Users\khali\PMN\Project_Final\Data\.bikeshops.csv"  
cloudera@localhost:/home/cloudera/Project/
```

```
scp -P 2222 "C:\Users\khali\PMN\Project_Final\Data\.orders.csv"  
cloudera@localhost:/home/cloudera/Project/
```

Pour le transfert des fichiers csv vers les répertoires créés dans HDFS respectivement :

```
hadoop fs -put /home/cloudera/Project/Customers.csv /user/cloudera/raw_data/Project/Customers  
hadoop fs -put /home/cloudera/Project/bikes.csv /user/cloudera/raw_data/Project/bikes  
hadoop fs -put /home/cloudera/Project/bikeshops.csv /user/cloudera/raw_data/Project/bikeshops  
hadoop fs -put /home/cloudera/Project/orders.csv /user/cloudera/raw_data/Project/orders
```

## Étape 2 : Exécution des Scripts PySpark

Un des problèmes rencontrés lors de l'exécution des scripts PySpark est l'absence de certaines dépendances, notamment les fichiers JAR nécessaires pour la lecture et l'écriture de fichiers CSV.

Pour résoudre cela, il était essentiel de télécharger les packages JAR appropriés (**spark-csv\_2.10** et **commons-csv**), puis de les placer dans le répertoire créé `/home/cloudera/spark-lib`.

### 1. Ajout fichiers JAR: Depuis terminal Windows machine local :

```
scp -P 2222 "C:\Users\khali\PMN\Project_Final\JAR\commons-csv-1.2.jar"  
cloudera@localhost:/home/cloudera/spark-lib/  
  
scp -P 2222 "C:\Users\khali\PMN\Project_Final\JAR\spark-csv_2.10-1.5.0.jar"  
cloudera@localhost:/home/cloudera/spark-lib/
```

Ensuite pour toute exécution des scripts py nécessitant ces dépendances, il était nécessaire d'ajouter l'emplacement dans la commande tel dans les commandes suivantes.

### 2. Nettoyage des Données :

#clean\_customers.py : Nettoie les données clients

```
spark-submit --jars /home/cloudera/spark-lib/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-lib/commons-csv-1.2.jar /home/cloudera/Project/clean_customers.py
```

#Clean\_customers2.py : Nettoie et formate les données clients

```
spark-submit --jars /home/cloudera/spark-libraries/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-libraries/commons-csv-1.2.jar /home/cloudera/Project/Clean_customers2.py
```

**#clean\_bikes.py** : Nettoie les données des vélos.

```
spark-submit --jars /home/cloudera/spark-libraries/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-libraries/commons-csv-1.2.jar /home/cloudera/Project/clean_bikes.py
```

**#clean\_bikeshops.py** : Nettoie les données des boutique des vélos.

```
spark-submit --jars /home/cloudera/spark-libraries/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-libraries/commons-csv-1.2.jar /home/cloudera/Project/clean_bikeshops.py
```

**#clean\_orders2.py** : Nettoie et formate les dates dans les données de commandes.

```
spark-submit --jars /home/cloudera/spark-libraries/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-libraries/commons-csv-1.2.jar /home/cloudera/Project/clean_orders2.py
```

**#create\_star\_schema2.py** : Crée la table de faits fact\_orders en joignant les dimensions clients et vélos. (boutiques vélos exclues par manque de lien logique)

```
spark-submit --jars /home/cloudera/spark-libraries/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-libraries/commons-csv-1.2.jar /home/cloudera/Project/create_star_schema2.py
```

**#create\_calendar\_lookup2.py** : Crée la table de lookup pour le calendrier.

```
spark-submit --jars /home/cloudera/spark-libraries/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-libraries/commons-csv-1.2.jar /home/cloudera/Project/create_calendar_lookup2.py
```

### 3. Configuration de cron pour l'Automatisation

Dans ce projet, cron est utilisé pour exécuter le script run\_cleaning\_pipeline.sh à intervalles réguliers afin de traiter et de mettre à jour automatiquement les données.

#### 1. Terminal cloudera:

```
cd Project
```

```
nano run_cleaning_pipeline.sh
```

```
chmod +x /home/cloudera/Project/run_cleaning_pipeline.sh
```

```
mkdir log
```

```
[cloudera@quickstart Project]$ nano /home/cloudera/Project/run_cleaning_pipeline.sh
[cloudera@quickstart Project]$ chmod +x /home/cloudera/Project/run_cleaning_pipeline.sh
[cloudera@quickstart Project]$ ls
bash: ls: command not found
[cloudera@quickstart Project]$ ls
bikes.csv  bikeshops.csv  clean_bikeshops.py  clean_bikes.py  clean_customers.py  clean_orders.py  create_star_schema.py  Customers.csv  orders.csv  run_cleaning_pipeline.sh
```

**Le script de run\_cleaning\_pipeline.sh:**

```
#!/bin/bash

# Directory containing the downloaded JAR files

JARS="/home/cloudera/spark-lib/spark-csv_2.10-1.5.0.jar,/home/cloudera/spark-lib/commons-csv-1.2.jar"

# Run each PySpark script with the necessary JARs

spark-submit --jars $JARS /home/cloudera/Project/clean_customers.py

spark-submit --jars $JARS /home/cloudera/Project/clean_orders.py

spark-submit --jars $JARS /home/cloudera/Project/clean_bikes.py

spark-submit --jars $JARS /home/cloudera/Project/clean_bikeshops.py

spark-submit --jars $JARS /home/cloudera/Project/create_star_schema.py
```

## 2. Ouvrir l'éditeur crontab :

```
crontab -e
```

## 3. Ajouter une ligne pour exécuter le script quotidiennement à minuit :

```
0 0 * * * /cloudera/home/run_cleaning_pipeline.sh >> /cloudera/home/Project/log/output.log 2>&1
```

Cette commande exécute le script run\_cleaning\_pipeline.sh chaque jour à minuit. La sortie et les erreurs sont redirigées vers un fichier de log pour faciliter le suivi et le débogage.

## 4. Vérifier la configuration de cron :

```
crontab -l
```

## Étape 3 : Création des Tables Hive

Après avoir nettoyé et transformé les données, j'ai créé les tables Hive depuis Hive Hue à partir des fichiers Parquet. Les codes suivants ont aussi inclus dans le répertoire Projet\_final.

### Création des tables:

```
use project_db;

drop table if exists customers_lookup;

CREATE EXTERNAL TABLE IF NOT EXISTS customers_lookup (

    CustomerKey STRING,

    Prefix STRING,

    FirstName STRING,

    LastName STRING,

    BirthDate STRING,

    MaritalStatus STRING,
```

```
Gender STRING,  
EmailAddress STRING,  
AnnualIncome STRING,  
TotalChildren STRING,  
EducationLevel STRING,  
Occupation STRING,  
HomeOwner STRING  
)  
  
STORED AS PARQUET LOCATION  
'hdfs://localhost/user/cloudera/modeled_data/Project/customers_lookup.parquet/';
```

```
drop table bikes_lookup;  
  
CREATE EXTERNAL TABLE IF NOT EXISTS bikes_lookup (  
    bike_id INT,  
    model STRING,  
    category1 STRING,  
    category2 STRING,  
    frame STRING,  
    price INT  
)  
  
STORED AS PARQUET LOCATION  
'hdfs://localhost/user/cloudera/modeled_data/Project/bikes_lookup.parquet/';
```

```
drop table bikeshops_lookup;  
  
CREATE EXTERNAL TABLE IF NOT EXISTS bikeshops_lookup (  
    bikeshop_id INT,  
    bikeshop_name STRING,  
    bikeshop_city STRING,  
    bikeshop_state STRING,  
    latitude STRING,  
    longitude STRING
```

```
)  
  
STORED AS PARQUET  
  
LOCATION 'hdfs://localhost/user/cloudera/modeled_data/Project/bikeshops_lookup.parquet/';
```

```
drop table fact_orders;  
  
CREATE EXTERNAL TABLE IF NOT EXISTS fact_orders (  
    OrderID INT,  
    OrderDate STRING,  
    CustomerID STRING,  
    FirstName STRING,  
    LastName STRING,  
    EmailAddress STRING,  
    BikeID INT,  
    model STRING,  
    category1 STRING,  
    category2 STRING,  
    frame STRING,  
    price INT,  
    quantity INT  
)  
  
STORED AS PARQUET  
  
LOCATION 'hdfs://localhost/user/cloudera/modeled_data/Project/fact_orders.parquet/';
```

```
SELECT * FROM project_db.fact_orders LIMIT 10;
```

#### Étape 4 : Querying des Tables Hive

Les commandes suivantes ont permis de comparer les resultat avec ce qui xistes dans les fichiers csv et d'identifier des erreurs corrigées dans le scripts précédents:

- **Type de donnée de "price"** : Initialement, la colonne price dans la table des bikes était définie comme un type float, mais Hive ne supportait pas ce format de manière adéquate. Pour résoudre cela, j'ai modifié le script clean\_bikes.py afin de transformer le type float en INT, permettant ainsi le traitement de cette table dans Hive.

- **Format des dates** : Les dates devaient être du même type entre les tables `calendar_lookup` et `fact_orders`.
  - Il était nécessaire d'utiliser un min/max des dates pour couvrir toute la période présente dans les fichiers CSV, afin d'inclure toutes les dates dans la `calendar_lookup_table`, assurant la cohérence et intégrité des données.
  - La transformation des dates devait aussi respecter un format cohérent afin de permettre la jointure des tables. Cette transformation a été réalisée dans le script `clean_orders.py`. Il s'agissait de nettoyer les dates, qui avaient des types et structures différentes dans la même colonne (par exemple, `m/dd/yyyy` était identifié en tant que `string`, et `mm/dd/yyyy` en tant que `date`). Les deux formats ont été uniformisés pour suivre le format standard de date `yyyy-MM-dd`, assurant ainsi une cohérence dans les jointures.

**"Checks"**

```
SELECT * FROM fact_orders LIMIT 10;
```

**"Checks"**

```
SELECT COUNT(*) AS null_order_id_count  
FROM fact_orders  
WHERE OrderID IS NULL;
```

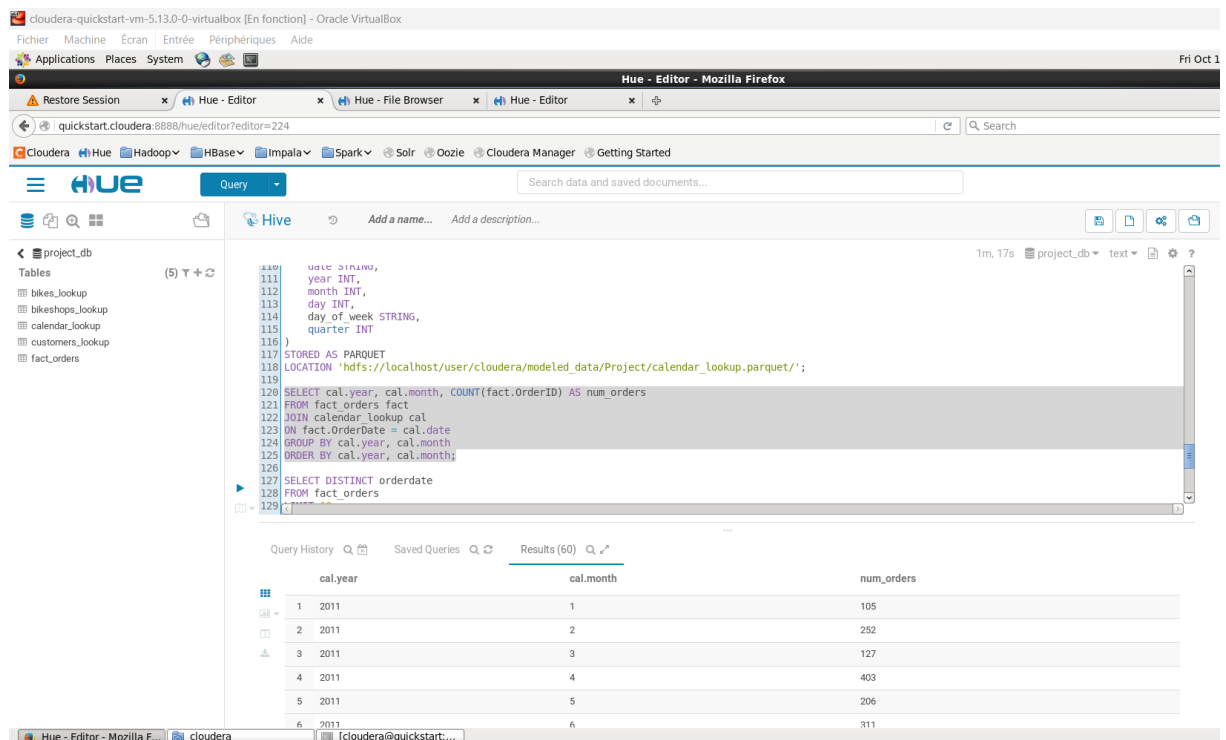
**"Monthly Sales Trend"**

```
SELECT cal.year, cal.month, COUNT(fact.OrderID) AS num_orders  
FROM fact_orders fact  
JOIN calendar_lookup cal  
ON fact.OrderDate = cal.date  
GROUP BY cal.year, cal.month  
ORDER BY cal.year, cal.month;
```



## PMN DAT25.1

### Mohamed Mustapha Abdennadher



The screenshot shows the Hue web interface in a Mozilla Firefox browser. The interface includes a top navigation bar with various Cloudera services like Hadoop, HBase, Impala, Spark, Solr, Oozie, and Cloudera Manager. The main area is divided into a left sidebar showing a file tree with 'project\_db' and its tables, and a central query editor. The query editor contains a Hive SQL query that joins 'fact\_orders' with 'calendar\_lookup' to calculate the number of orders per year and month. Below the query editor, the results are displayed in a table with 6 rows and 3 columns: cal.year, cal.month, and num\_orders.

cal.year	cal.month	num_orders
1	2011	105
2	2011	252
3	2011	127
4	2011	403
5	2011	206
6	2011	311

#### "Total Sales per Bike Model"

```
SELECT model, SUM(quantity) AS total_sales
FROM fact_orders
GROUP BY model
ORDER BY total_sales DESC
LIMIT 10;
```

#### "Total Sales by Customer"

```
SELECT FirstName, LastName, SUM(quantity) AS total_purchases
FROM fact_orders
GROUP BY FirstName, LastName
ORDER BY total_purchases DESC
LIMIT 10;
```

#### "Monthly Sales Trend"

```
SELECT cal.year, cal.month, COUNT(fact.OrderID) AS num_orders
FROM fact_orders fact
JOIN calendar_lookup cal
```

PMN DAT25.1  
Mohamed Mustapha Abdennadher

```
ON fact.OrderDate = cal.date  
  
GROUP BY cal.year, cal.month  
  
ORDER BY cal.year, cal.month;
```

#### **"Total Sales by Category and Model"**

```
SELECT category1, model, SUM(quantity) AS total_sales  
  
FROM fact_orders  
  
GROUP BY category1, model  
  
ORDER BY total_sales DESC  
  
LIMIT 10;
```

#### **"Highest Priced Bikes Sold"**

```
SELECT model, price, SUM(quantity) AS total_sold  
  
FROM fact_orders  
  
GROUP BY model, price  
  
ORDER BY price DESC, total_sold DESC  
  
LIMIT 10;
```

#### **"Customer Purchase History"**

```
SELECT FirstName, LastName, model, SUM(quantity) AS total_bikes_purchased  
  
FROM fact_orders  
  
GROUP BY FirstName, LastName, model  
  
ORDER BY total_bikes_purchased DESC  
  
LIMIT 10;
```

#### **"Sales by Frame Type"**

```
SELECT frame, SUM(quantity) AS total_sales  
  
FROM fact_orders  
  
GROUP BY frame  
  
ORDER BY total_sales DESC;
```

#### "Top Selling Bike Categories"

```
SELECT category1, SUM(quantity) AS total_sales
FROM fact_orders
GROUP BY category1
ORDER BY total_sales DESC
LIMIT 10;
```

#### "Export Query Results to HDFS in CSV Format"

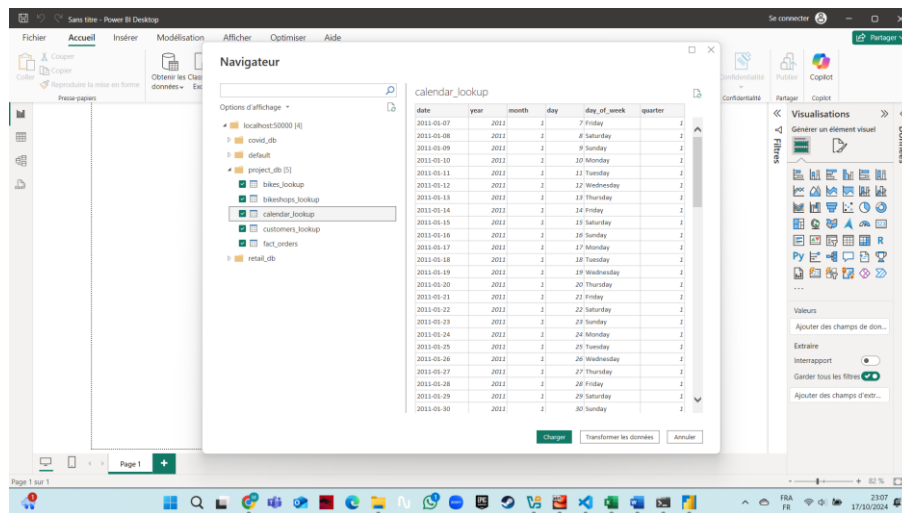
```
INSERT OVERWRITE DIRECTORY '/user/cloudera/output/sales_report_csv'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT model, SUM(quantity) AS total_sales
FROM fact_orders
GROUP BY model;
```

#### Étape 4 : Connexion à Power BI

Connexion à la project\_db Impala utilisant le connecteur ODBC de Power BI.

##### Détails de connexion :

- **Serveur** : localhost
- **Port** : 50000
- **Port client**: 21050
- **Base de données** : project\_db



## Création de la relation dans Power BI

Nous avons établi une relation entre la table de faits (*fact\_orders*) et la table de recherche de calendrier (*calendar\_lookup*) pour permettre des analyses basées sur le temps.

### Table source : *fact\_orders*

- Colonne liée : orderdate

### Table de destination : *calendar\_lookup*

- Colonne liée : date

### Choix de la cardinalité :

Nous avons sélectionné une **cardinalité de type plusieurs à un (\*:1)**, car chaque date dans la table de faits peut correspondre à une seule date dans la table de calendrier, mais une date du calendrier peut être liée à plusieurs enregistrements dans la table de faits.

### Direction du filtre croisé :

Nous avons sélectionné **À sens unique**, afin que les filtres appliqués sur la table *calendar\_lookup* impactent les résultats de la table *fact\_orders*.

Cette relation permet de générer des analyses basées sur le temps (par exemple, les ventes par mois, par trimestre, ou par année), en associant les dates des commandes avec les dates disponibles dans le calendrier.

## Aperçu des Données et Analyses

Les dimensions (clients, vélos, magasins) et la table des faits (orders) permettent d'analyser les ventes par périodes et par type de vélo. Power BI peut être utilisé pour visualiser ces tendances à l'aide des filtres de calendrier.

## Overview



tableau prêt à être exporté ou filtrés encore plus sur une années précise ou un mois grâce aux filtres mis à disposition.

