**Banking Data Analysis using hadoop**

L'objectif principal est de démontrer la manière dont les données brutes, stockées initialement dans une base de données relationnelle MySQL, peuvent être transférées, sécurisées et analysées dans un écosystème Big Data. Ce processus met en œuvre des techniques avancées, notamment l’intégration des données dans HDFS (Hadoop Distributed File System), la création de tables analytiques dans Hive, et l’utilisation de fonctions de chiffrement pour garantir la confidentialité des informations.

1. **Chargement des données dans mysql :**

**Entrée dans le shell mysql**

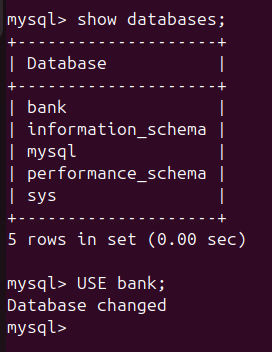
mysql -u root -p



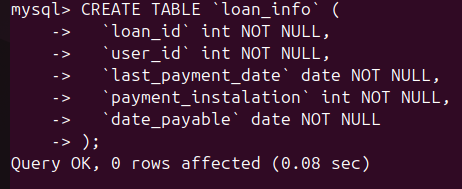
**Création de base de donnée bank dans mysql**

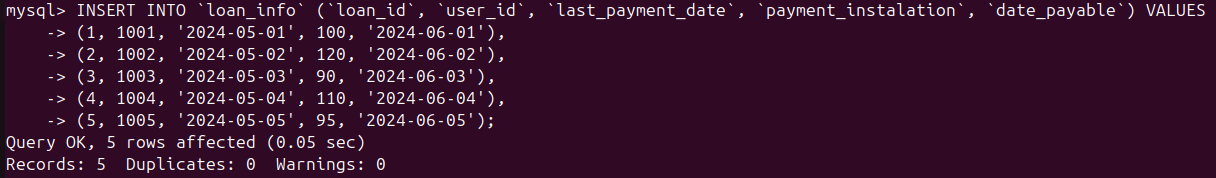
CREATE DATABASE bank;

USE bank;

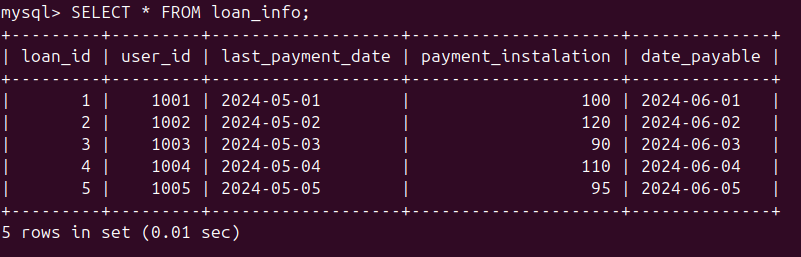


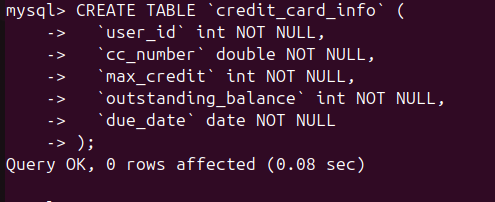
**Créer des tables dans mysql et insérer les données dans les tables mysql**



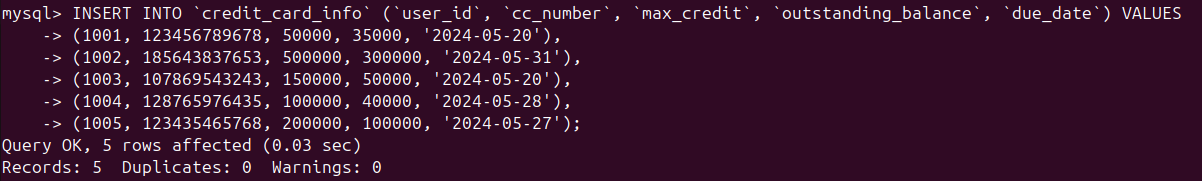


**Vérification des données dans le tableau d'informations sur les prêts**

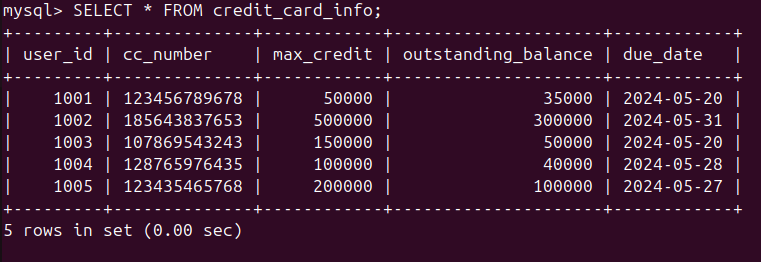


**Création de la table credit\_card\_info**

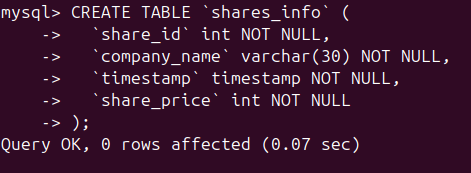
**Insertion de données dans la table credit\_card\_info**



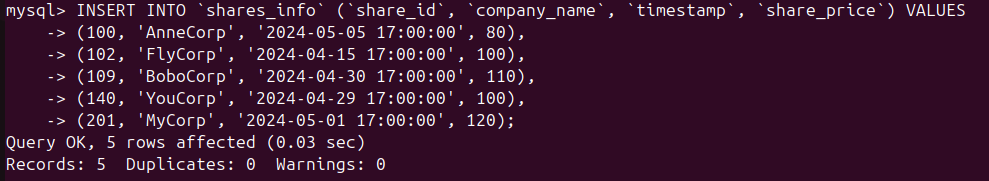
**Vérification des données dans le tableau d'informations de carte de credit**



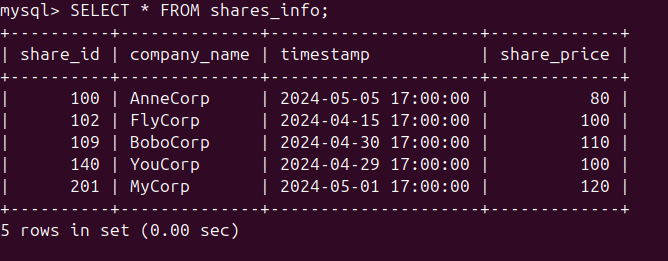
**Création de la table shares\_info**



**Insertion de données dans la table shares\_info**



**Vérification des données dans le tableau d'informations sur les actions**



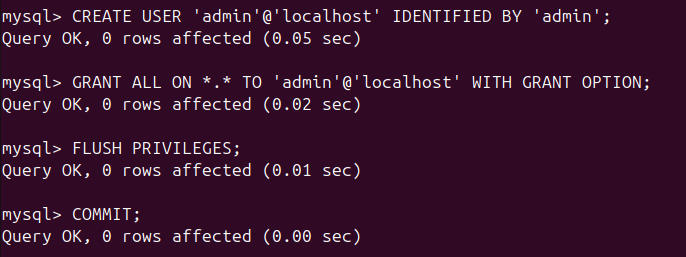
COMMIT;

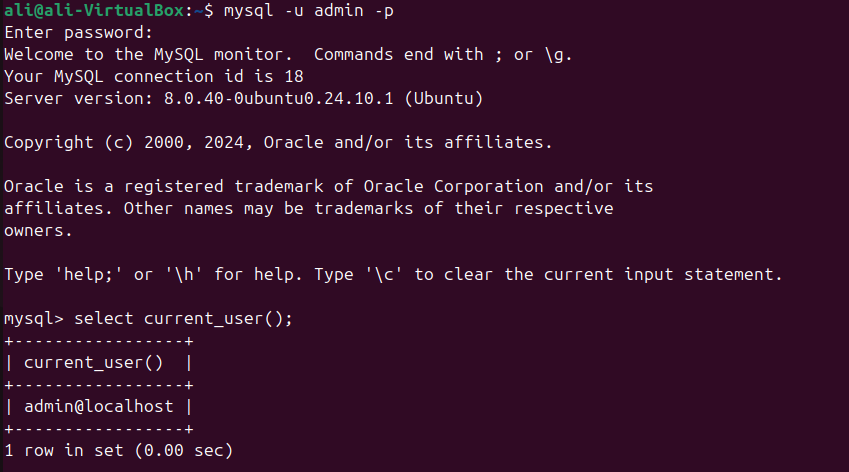


1. **Exporter des données de Mysql vers HDFS en utilisant sqoop**

Nous avons maintenant des données dans mysql. Nous devons exporter ces données vers HDFS. Nous le ferons en utilisant sqoop.

Pour exporter des données vers HDFS, nous allons d'abord créer un administrateur dans mysql.





Transférons maintenant ces tables dans HDFS en écrivant des tâches sqoop.

**Sqoop job pour transférer les données de la table loan\_info**

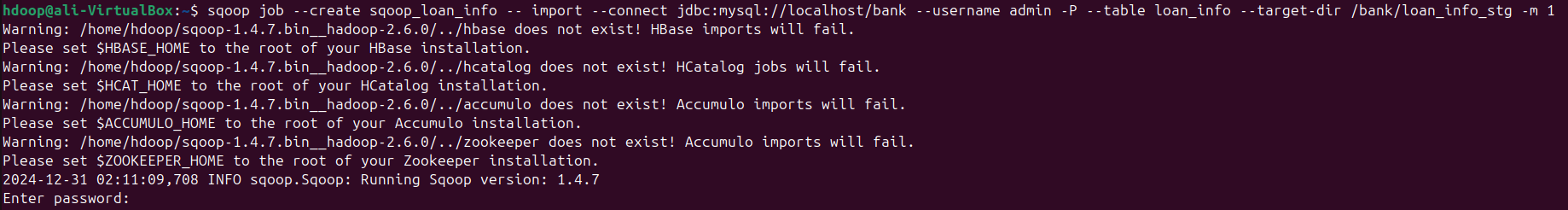
**Créer un répertoire dans HDFS loan\_info\_stg pour stocker les données de la table**

hadoop fs -mkdir /bank

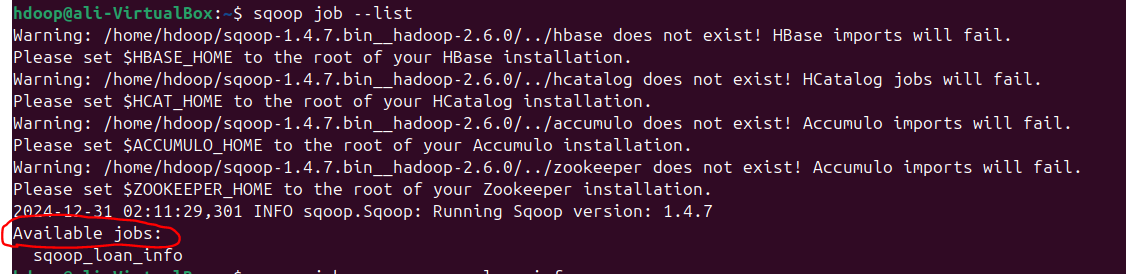
hadoop fs -mkdir /bank/loan\_info\_stg

**Création d'une tâche sqoop pour sqoop\_loan\_info**

**sqoop job --create sqoop\_loan\_info -- import --connect jdbc:mysql://localhost/bank --username admin -P --table loan\_info --target-dir /bank/loan\_info\_stg -m 1**

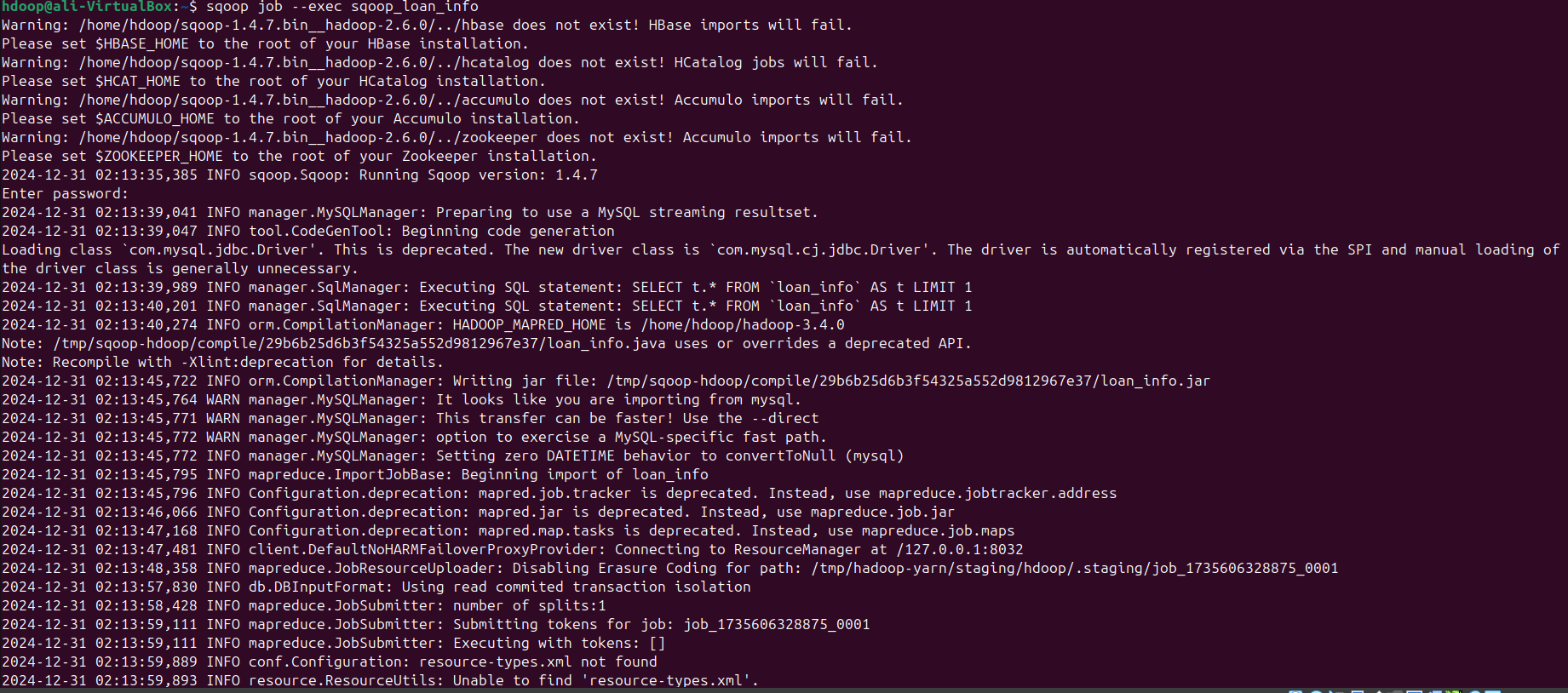


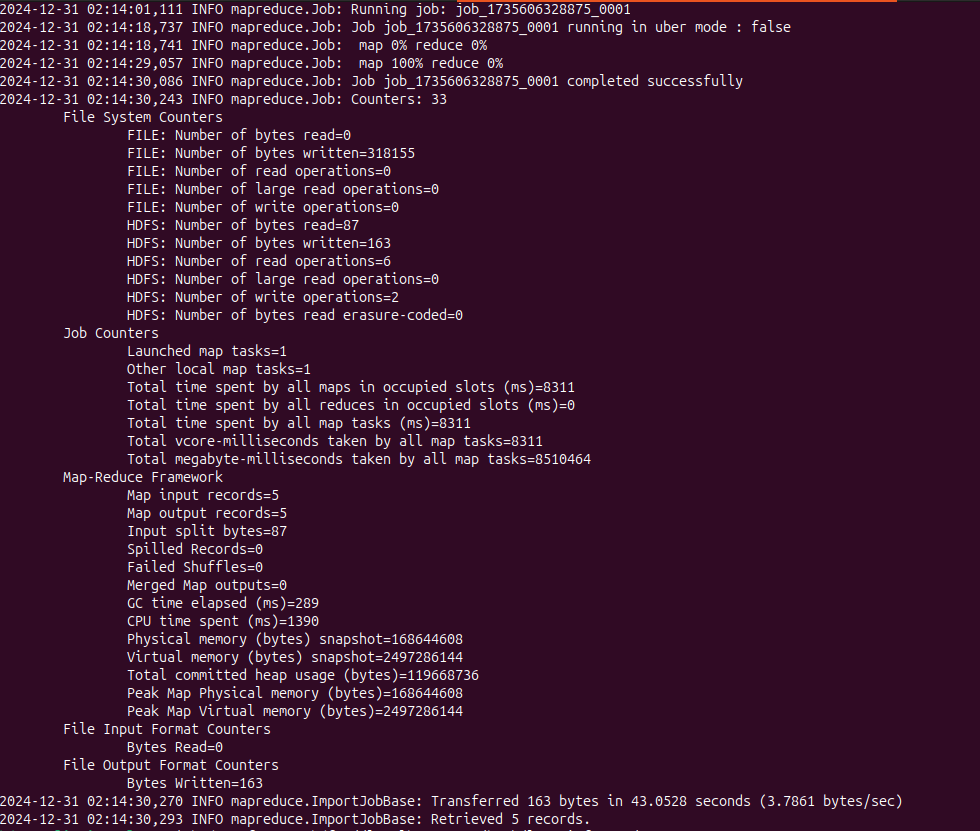
**sqoop job --list**

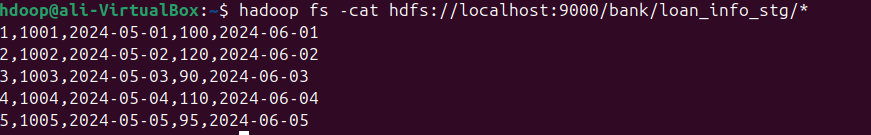


**Exécution de la tâche sqoop pour sqoop\_loan\_info**

**sqoop job –exec sqoop\_loan-info**





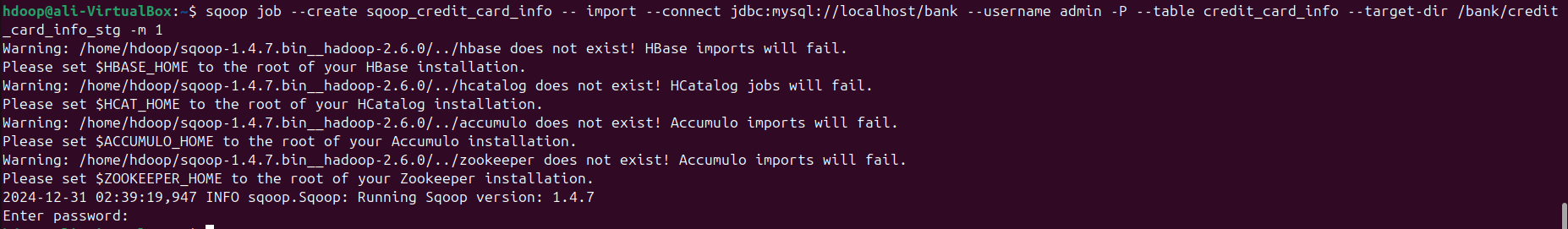


**Créez un répertoire dans HDFS credit\_card\_info\_stg pour stocker les données de la table**

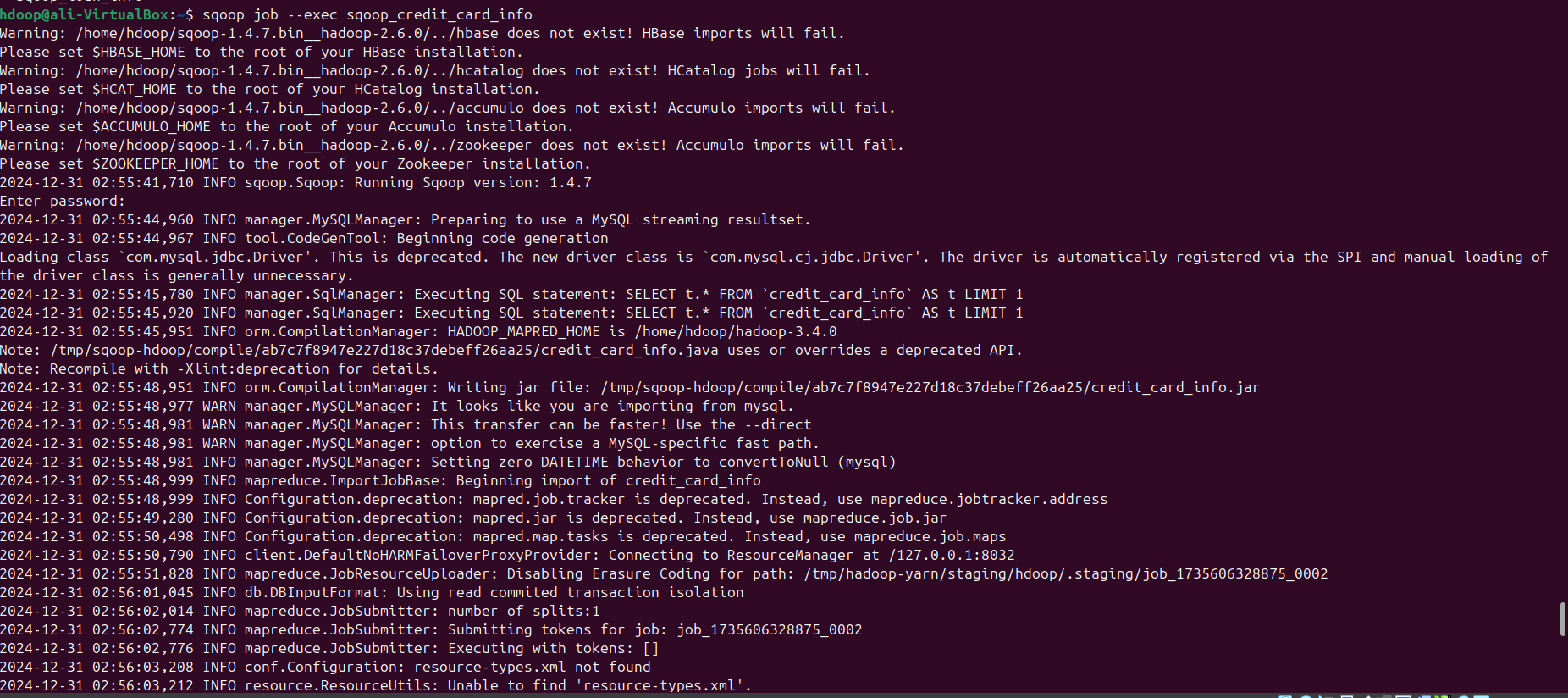
hadoop fs -mkdir /bank/credit\_card\_info\_stg

**Création d'une tâche sqoop pour sqoop\_credit\_card \_info**

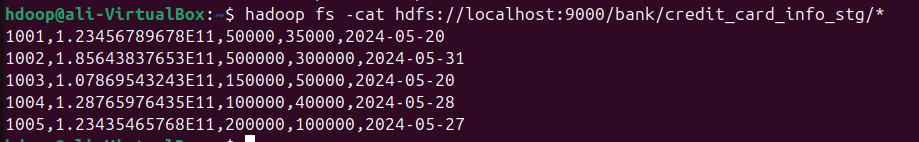
**sqoop job --create sqoop\_credit\_card\_info -- import --connect jdbc:mysql://localhost/bank --username admin -P --table credit\_card\_info --target-dir /bank/credit\_card\_info\_stg -m 1**



**sqoop job --exec sqoop\_credit\_card\_info**



**hadoop fs -cat hdfs://localhost:9000/bank/credit\_card\_info\_stg/\***

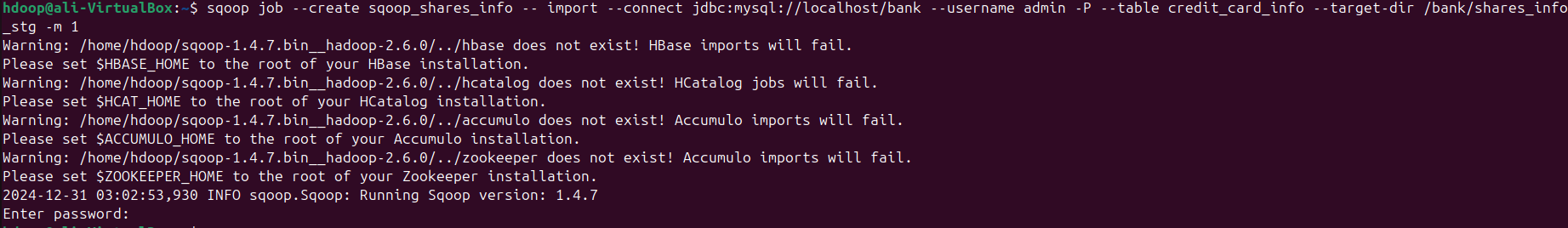


**Créez un répertoire dans HDFS shares\_info\_stg pour stocker les données de la table**

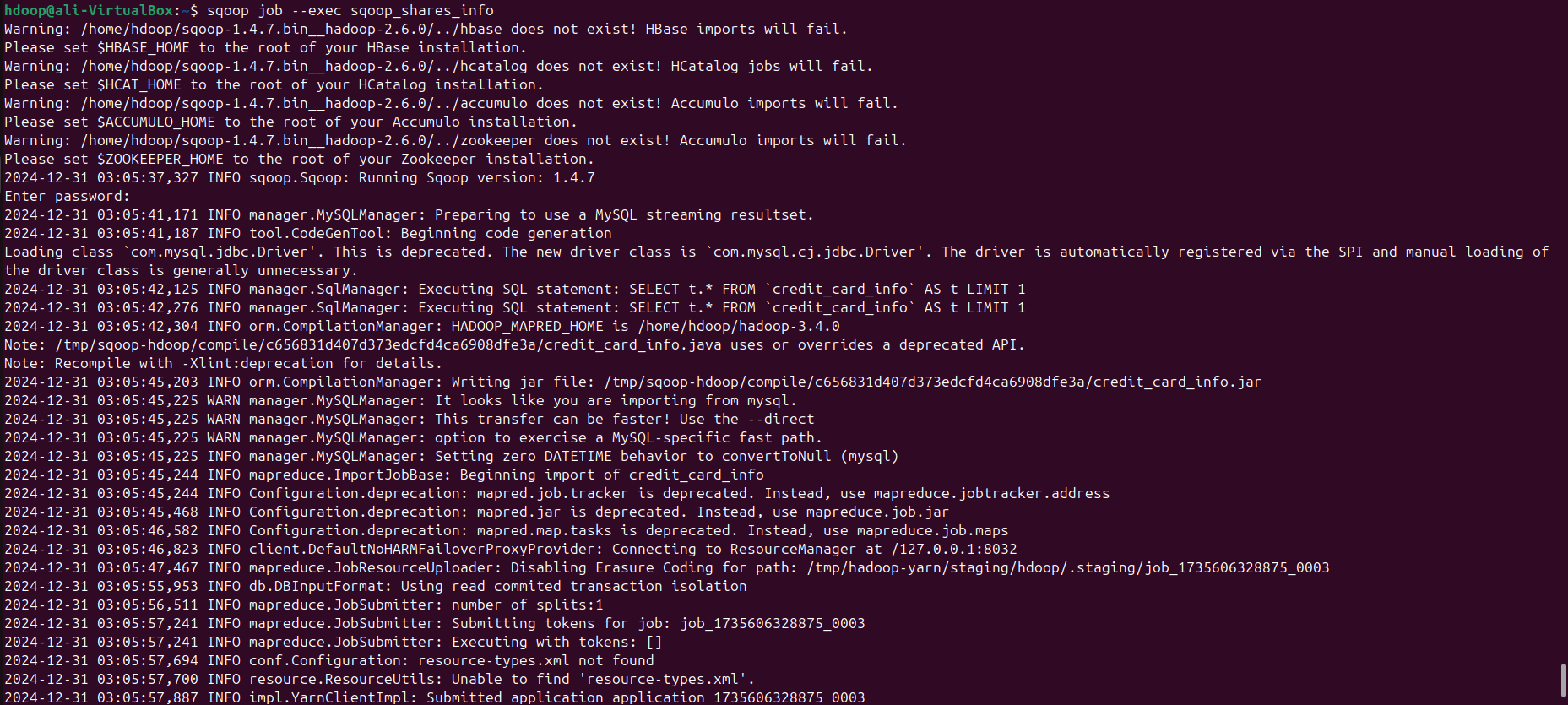
hadoop fs -mkdir /bank/shares\_info\_stg

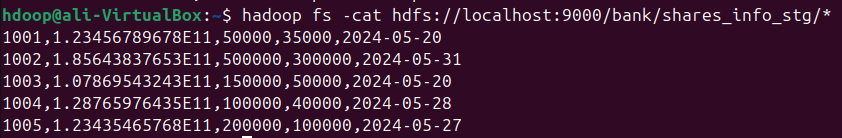
**Création d'une tâche sqoop pour sqoop\_shares \_info**

**sqoop job --create sqoop\_shares\_info -- import --connect jdbc:mysql://localhost/bank --username admin -P --table credit\_card\_info --target-dir /bank/shares\_info\_stg -m 1**

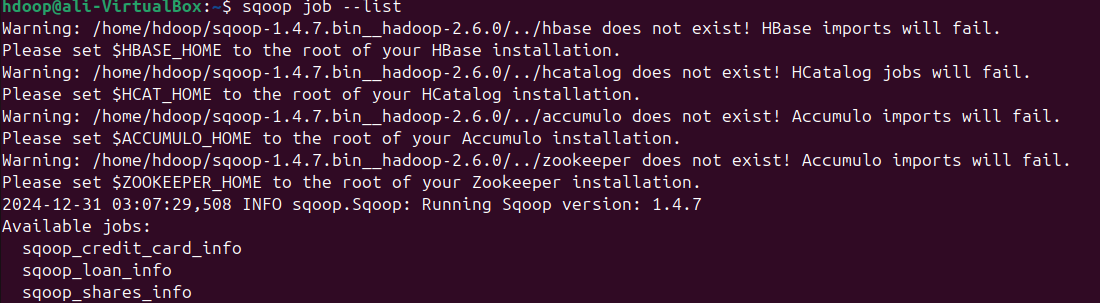


**sqoop job --exec sqoop\_shares\_info**



**hadoop fs -cat hdfs://localhost:9000/bank/shares\_info\_stg/\*** 

**sqoop job --list**

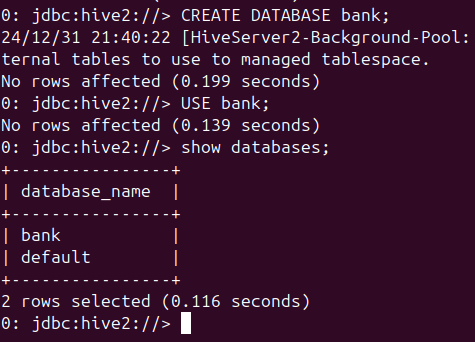


1. **Créer des tables externes dans Hive**

**Create database**

CREATE DATABASE bank;

USE bank;



**Création de la table loan\_info\_stg**

Comme cette table est une table externe, nous devons simplement donner l’emplacement des données.

CREATE EXTERNAL TABLE loan\_info\_stg (

loan\_id int,

user\_id int,

last\_payment\_date string,

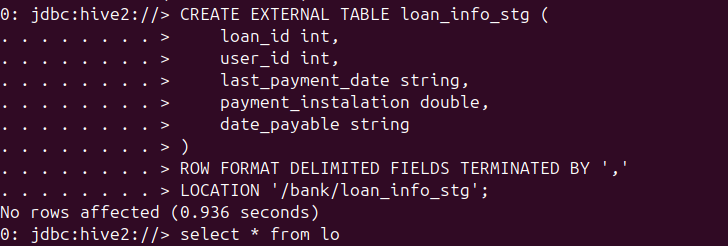
payment\_instalation double,

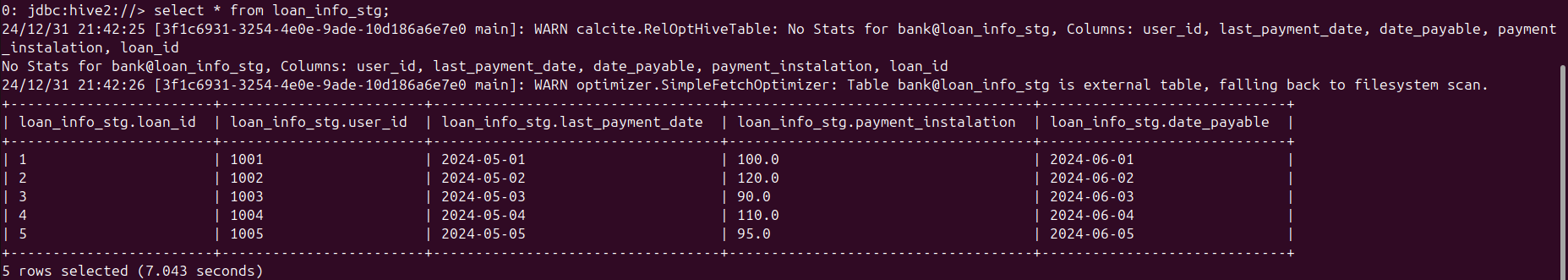
date\_payable string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LOCATION '/bank/loan\_info\_stg';





**Création de la table credit\_card\_info\_stg**

CREATE EXTERNAL TABLE credit\_card\_info\_stg (

user\_id int,

cc\_number string,

max\_credit double,

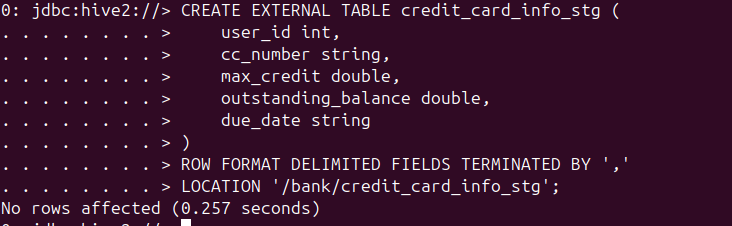
outstanding\_balance double,

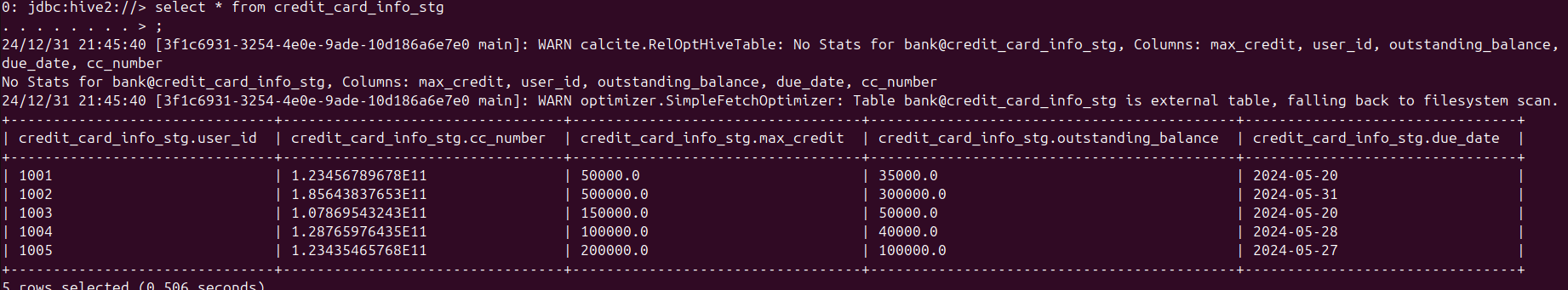
due\_date string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LOCATION '/bank/credit\_card\_info\_stg';





**Création de la table shares\_info\_stg**

CREATE EXTERNAL TABLE shares\_info\_stg (

share\_id string,

company\_name string,

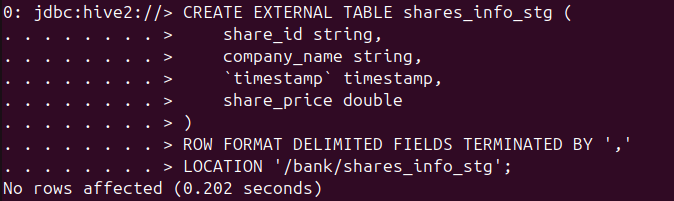
`timestamp` timestamp,

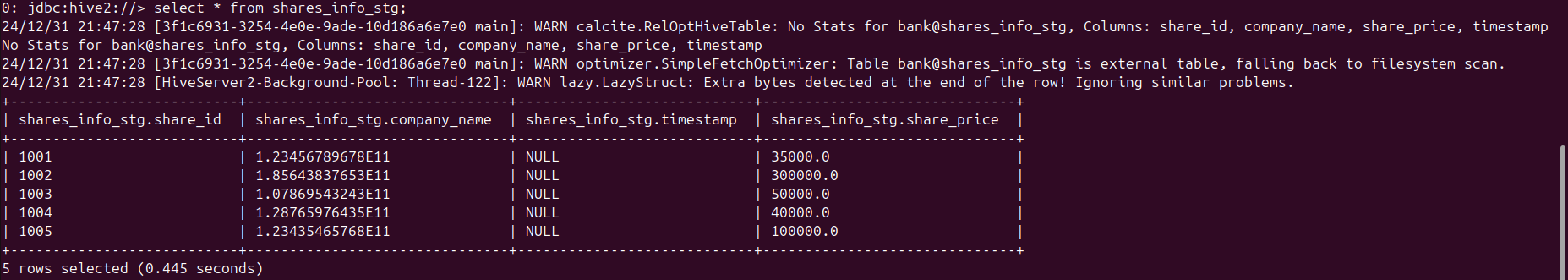
share\_price double

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

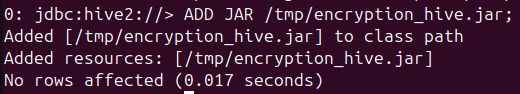
LOCATION '/bank/shares\_info\_stg';

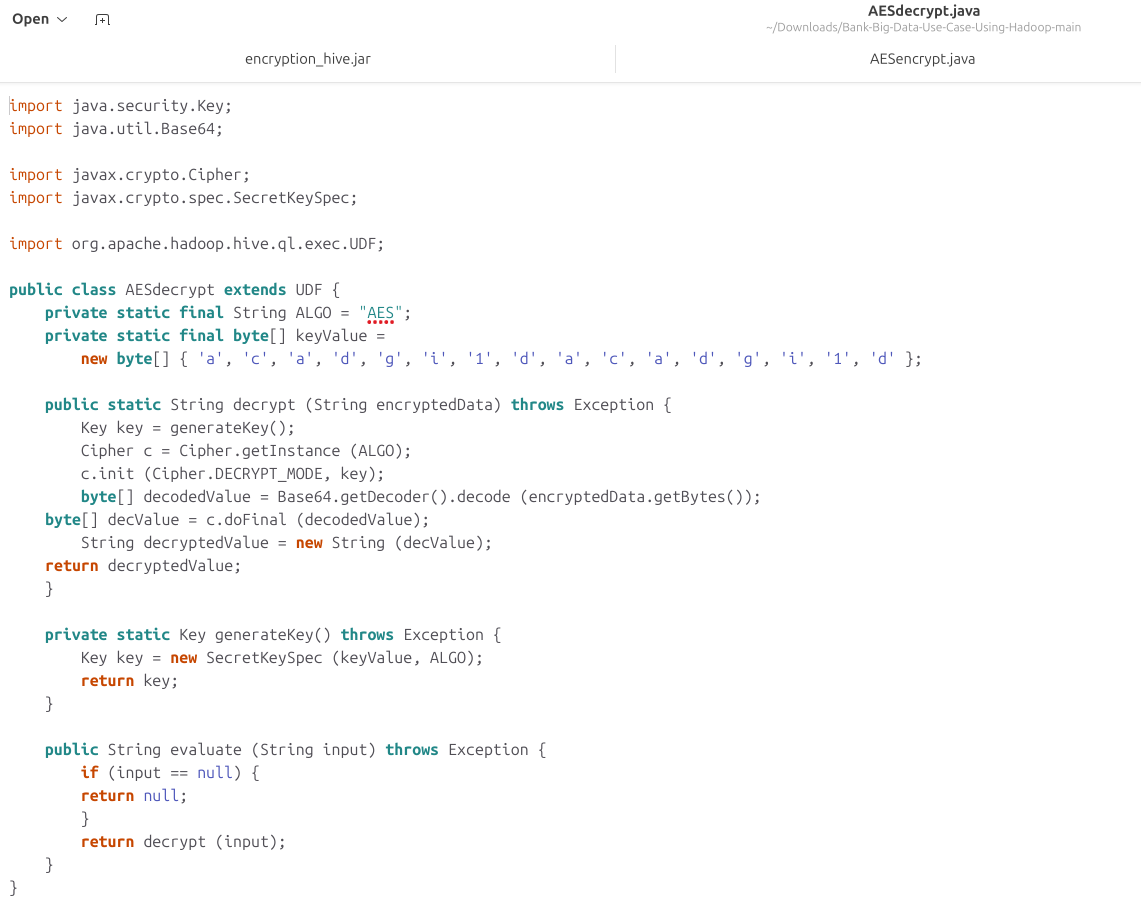




**Création de tables principales et chargement des données dans les tables principales à partir des tables stg :**

Ajoute du cryptage dans le shell de Hive :







Les deux classes **AESdecrypt.java** et **AESencrypt.java** sont des fonctions définies par l'utilisateur (UDF) pour **Apache Hive**, permettant de chiffrer et déchiffrer des données en utilisant l'algorithme AES avec une clé de 128 bits.

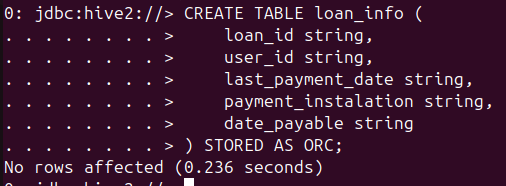
**Intégration avec Hive :**

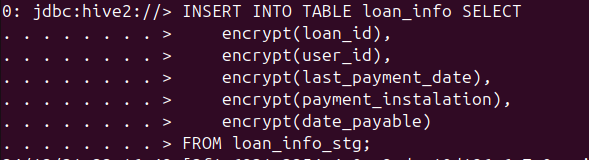
**Enregistrement des fonctions** : Après avoir compilé les classes Java, vous devez les ajouter à Hive via **ADD JAR** et déclarer les fonctions avec **CREATE FUNCTION**.

**CREATE TEMPORARY FUNCTION encrypt AS 'AESencrypt';**

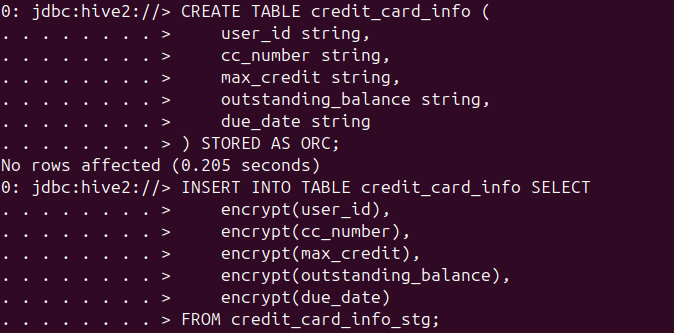
**CREATE TEMPORARY FUNCTION decrypt AS 'AESdecrypt';**

**Création de la table loan\_info et insertion des données à crypter :**

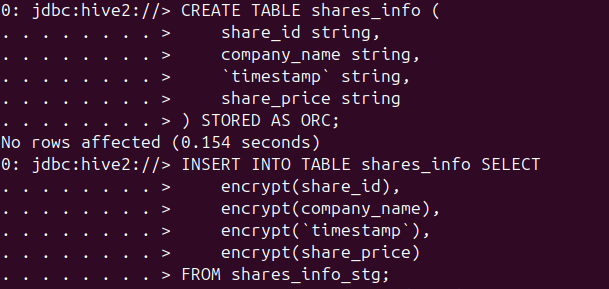




**Création de la table credit\_card\_info et insertion des données à crypter :**



**Création de la table shares\_info et insertion des données à crypter :**



**Vérification des données dans les 3 tableaux :**

Comme les données sont bancaires, nous avons crypté les données.



**Pour décrypter les données, vous pouvez utiliser :**

**SELECT**

**decrypt(loan\_id),**

**decrypt(user\_id),**

**decrypt(last\_payment\_date),**

**decrypt(payment\_instalation),**

**decrypt(date\_payable)**

**FROM loan\_info;**

**SELECT**

**decrypt(user\_id),**

**decrypt(cc\_number),**

**decrypt(max\_credit),**

**decrypt(outstanding\_balance),**

**decrypt(due\_date)**

**FROM credit\_card\_info;**

**SELECT**

**decrypt(share\_id),**

**decrypt(company\_name),**

**decrypt(`timestamp`),**

**decrypt(share\_price)**

**FROM shares\_info;**

1. **Analysis**

* **Loan\_info table problem statements**

Decrypting the data for analysis

1-

Cette commande récupère la valeur user\_id de la table Loan\_info après avoir déchiffré la colonne last\_payment\_date,

et sélectionnez uniquement les lignes dont la différence de date est supérieure ou égale à 20 jours.

**Solution :**

**SELECT decrypt(user\_id)**

**FROM loan\_info**

**WHERE datediff(from\_unixtime(unix\_timestamp(), 'yyyy-MM-dd'), decrypt(last\_payment\_date)) >= 20;**

2-

La commande SQL ci-dessus prend le user\_id de la table Loan\_info qui est jointe à la table credit\_card\_info

basé sur l'id\_utilisateur déchiffré. Les résultats incluent uniquement les données pour lesquelles le solde\_encours est activé.

la table credit\_card\_info est 300000.0 ou plus, et la date last\_payment\_date dans la table Loan\_info survient 15 jours

ou plus avant la date du jour.

**Solution :**

**SELECT decrypt(li.user\_id)**

**FROM loan\_info li INNER JOIN credit\_card\_info cci**

**ON decrypt(li.user\_id) = decrypt(cci.user\_id)**

**WHERE CAST(decrypt(cci.outstanding\_balance) AS double) >= 300000.0**

**AND datediff(from\_unixtime(unix\_timestamp(), 'yyyy-MM-dd'), decrypt(li.last\_payment\_date))>=15;**

3-

Cette commande récupère les données de la table share\_info après avoir déchiffré certaines colonnes et

sélectionnez uniquement les lignes où share\_id est égal ou supérieur à 400.

**Solution :**

**SELECT decrypt(share\_id), decrypt(company\_name), decrypt(`timestamp`), decrypt(share\_price)**

**FROM shares\_info WHERE decrypt(share\_id) >= 400;**