

Tutoriel Hive
Commandes de Base, Création de Tables et Optimisations

Mohamed KOUBAA

2025/2026

Contents

1	Introduction	4
2	Opérations sur les bases de données	4
2.1	Créer une base de données	4
2.2	Lister les bases existantes	4
2.3	Utiliser une base	4
2.4	Supprimer une base	4
3	Création de tables : syntaxe générale	4
4	Organisation des bases et tables dans HDFS	5
5	Exemples de création de tables	5
5.1	Table simple	5
5.2	Table EXTERNAL	6
5.3	Définir un format personnalisé (ROW FORMAT)	6
5.4	Spécifier un format de stockage (STORED AS)	6
5.5	Stockage à un emplacement spécifique (LOCATION)	6
5.6	Créer une table à partir d'une requête (CTAS)	7
6	SerDe : lire et écrire les fichiers	7
6.1	SerDe par défaut dans Hive	7
6.2	CSV avec OpenCSVSerde	7
6.3	Regex SerDe	8
6.4	Serde automatiques selon le format de stockage	8
6.5	TBLPROPERTIES : propriétés de la table	8
7	Insérer des données	8
7.1	Insertion directe	8
7.2	Insertion depuis SELECT	9
8	Voir la structure d'une table	9
9	Chargement de données avec LOAD DATA	9
9.1	Chargement depuis HDFS	9
9.2	Chargement depuis le système local	9
9.3	Remplacer les données	9
10	Conversion d'un fichier texte vers Parquet (via table temporaire)	9
11	Partitionnement (PARTITIONED BY)	10
11.1	Chargement dynamique	10
12	Table partitionnée en mode statique	10
12.1	Chargement statique	11
13	Bucketing (CLUSTERED BY)	11

14 JOIN optimisé avec des tables bucketées	11
14.1 Préparation	11
14.2 JOIN optimisé	12
15 Conclusion	12

1 Introduction

Hive est un outil de data warehousing pour Hadoop permettant d'interroger de grandes volumes de données via un langage similaire à SQL, appelé HiveQL. Ce tutoriel introduit les commandes essentielles pour manipuler des bases, créer des tables et gérer divers types de stockage. Les concepts avancés comme le partitionnement et le bucketing sont présentés à la fin, accompagnés d'un exemple de JOIN optimisé avec buckets.

2 Opérations sur les bases de données

2.1 Crée une base de données

```
CREATE DATABASE IF NOT EXISTS ventes_db;
```

Explication : Cette commande crée une base de données nommée `ventes_db`. L'option `IF NOT EXISTS` empêche une erreur si la base existe déjà.

2.2 Lister les bases existantes

```
SHOW DATABASES;
```

Explication : Affiche toutes les bases disponibles dans Hive.

2.3 Utiliser une base

```
USE ventes_db;
```

Explication : Définit la base courante dans laquelle toutes les opérations suivantes seront exécutées.

2.4 Supprimer une base

```
DROP DATABASE ventes_db;
DROP DATABASE ventes_db CASCADE;      -- supprime aussi les tables
```

Explication : Par défaut, la base ne peut pas être supprimée si elle contient des tables. `CASCADE` permet de tout supprimer.

3 Création de tables : syntaxe générale

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
[db.]table_name (
    col_name data_type [COMMENT description],
    ...
)
[COMMENT table_comment]
[ROW FORMAT ...]
[STORED AS file_format]
[LOCATION 'hdfs_path']
[AS select_statement];
```

- **TEMPORARY** : table visible uniquement dans la session en cours.
- **EXTERNAL** : Hive ne gère pas physiquement les données (ne les supprime pas).
- **STORED AS** : définit le format (TEXTFILE, PARQUET, ORC...).
- **LOCATION** : définit un chemin HDFS personnalisé.
- **AS SELECT** : crée une table directement à partir d'une requête.

4 Organisation des bases et tables dans HDFS

Lorsqu'une base de données Hive est créée, Hive génère automatiquement un répertoire dans HDFS portant le même nom que la base, avec l'extension .db. Par exemple :

```
CREATE DATABASE ventes_db;
```

Hive crée le répertoire suivant :

```
/user/hive/warehouse/ventes_db.db
```

Explication : Le suffixe .db permet de distinguer les bases des tables dans le répertoire warehouse.

À l'intérieur d'une base, chaque table créée produit un répertoire dont le nom correspond exactement à celui de la table, sans suffixe :

```
CREATE TABLE produits (id INT, nom STRING, prix DOUBLE);
```

Hive crée automatiquement :

```
/user/hive/warehouse/ventes_db.db/produits
```

Résumé :

- Base Hive → dossier `ma_base.db`
- Table Hive → dossier `table_name` dans la base

5 Exemples de création de tables

5.1 Table simple

```
CREATE TABLE produits (
    id INT COMMENT 'Identifiant produit',
    nom STRING,
    prix DOUBLE
)
COMMENT 'Table des produits';
```

Explication : Table interne (gérée par Hive), stockée par défaut en TEXTFILE.

5.2 Table EXTERNAL

```
CREATE EXTERNAL TABLE logs_web (
    ip STRING,
    url STRING,
    ts STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/data/logs/web';
```

Pourquoi EXTERNAL ? Si l'on supprime la table Hive, les fichiers dans /data/logs/web restent intacts.

5.3 Définir un format personnalisé (ROW FORMAT)

```
CREATE TABLE utilisateurs (
    id INT,
    nom STRING,
    pays STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

Explication : Hive peut lire des données délimitées grâce aux options FIELDS et LINES.

5.4 Spécifier un format de stockage (STORED AS)

```
CREATE TABLE ventes_parquet (
    id INT,
    produit STRING,
    montant DOUBLE
)
STORED AS PARQUET;
```

Explication : Les formats colonnes comme PARQUET ou ORC améliorent la compression et les performances.

5.5 Stockage à un emplacement spécifique (LOCATION)

```
CREATE EXTERNAL TABLE catalogue (
    id INT,
    titre STRING
)
LOCATION '/datasets/catalogue';
```

Explication : Hive lit les fichiers déjà présents dans ce répertoire HDFS.

5.6 Crée une table à partir d'une requête (CTAS)

```
CREATE TABLE ventes_2023 AS  
SELECT *  
FROM ventes  
WHERE annee = 2023;
```

Explication : La table est créée et immédiatement remplie avec le résultat de la requête.

6 SerDe : lire et écrire les fichiers

Une SerDe (*Serializer/Deserializer*) permet à Hive de convertir les lignes d'un fichier en colonnes exploitables, puis de réécrire les données si nécessaire. Chaque table Hive utilise une SerDe pour interpréter son contenu.

6.1 SerDe par défaut dans Hive

Lorsque l'on crée une table sans préciser explicitement ROW FORMAT SERDE, Hive utilise automatiquement :

```
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
```

Caractéristiques du SerDe par défaut :

- Format de données : **TEXTFILE**.
- Séparateur de colonnes : **tabulation** (\t).
- Valeurs nulles représentées par : \N.
- Lecture “lazy” : les champs ne sont convertis en types Hive qu'au moment de leur utilisation.
- Pas de gestion d'échappement avancée.
- Convient aux fichiers texte simples, sans structure complexe.

Exemple utilisant le SerDe par défaut :

```
CREATE TABLE default_format (  
    id INT,  
    name STRING,  
    salary DOUBLE  
) ;  
-- Ici LazySimpleSerDe est utilisé automatiquement
```

6.2 CSV avec OpenCSVSerde

```
CREATE TABLE students (  
    name STRING,  
    gender STRING,  
    moy DECIMAL(4,2)  
)  
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';
```

6.3 Regex SerDe

```
CREATE TABLE logs_parsed (
    ip STRING,
    ts STRING,
    request STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    "input.regex" = "(\\S+)\\s+(\\S+)\\s+\"(.*)\""
);
```

6.4 SerDe automatiques selon le format de stockage

Certains formats choisissent leur propre SerDe :

- ORC : OrcSerde
- Parquet : ParquetHiveSerDe
- Avro : AvroSerDe

6.5 TBLPROPERTIES : propriétés de la table

Exemple d'utilisation pour gérer :

- présence d'un header ;
- valeurs nulles personnalisées.

```
CREATE TABLE sales_csv (
    id INT,
    product STRING,
    amount DOUBLE
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
TBLPROPERTIES (
    "skip.header.line.count"="1",
    "serialization.null.format"="NULL"
);
```

7 Insérer des données

7.1 Insertion directe

```
INSERT INTO TABLE produits VALUES
(1, 'Laptop', 2500),
(2, 'Smartphone', 900);
```

7.2 Insertion depuis SELECT

```
INSERT INTO TABLE ventes
SELECT * FROM ventes_staging;
```

8 Voir la structure d'une table

```
DESCRIBE FORMATTED produits;
```

Explication : Affiche toutes les métadonnées (schéma, format, chemin HDFS, partitions...).

9 Chargement de données avec LOAD DATA

9.1 Chargement depuis HDFS

```
LOAD DATA INPATH '/data/employees.csv'
INTO TABLE employees;
```

9.2 Chargement depuis le système local

```
LOAD DATA LOCAL INPATH '/home/user/data.txt'
INTO TABLE sales_csv;
```

9.3 Remplacer les données

```
LOAD DATA INPATH '/data/new.csv'
OVERWRITE INTO TABLE employees;
```

10 Conversion d'un fichier texte vers Parquet (via table temporaire)

Étape 1 : table temporaire texte

```
CREATE TEMPORARY TABLE tmp_sales (
    id INT,
    product STRING,
    price DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Étape 2 : chargement du fichier local

```
LOAD DATA LOCAL INPATH '/home/user/sales.txt'
INTO TABLE tmp_sales;
```

Étape 3 : table finale au format Parquet

```
CREATE TABLE sales_parquet (
    id INT,
    product STRING,
    price DOUBLE
)
STORED AS PARQUET;
```

Étape 4 : conversion

```
INSERT INTO TABLE sales_parquet
SELECT * FROM tmp_sales;
```

11 Partitionnement (PARTITIONED BY)

Rappel : Les partitions organisent les données physiquement par répertoires HDFS, améliorant les recherches sur les colonnes filtrées.

```
CREATE TABLE ventesPart (
    id INT,
    produit STRING,
    quantite INT
)
PARTITIONED BY (annee INT, mois INT)
STORED AS PARQUET;
```

Avantage : Les requêtes avec WHERE annee=2023 ne lisent que les partitions utiles.

11.1 Chargement dynamique

```
set hive.exec.dynamic.partition.mode=nonstrict;

insert into ventes partition(annee, mois)
select id,produit,quantite,annee,mois from ventes;
```

12 Table partitionnée en mode statique

```
CREATE TABLE ventesStatic(
    id INT,
    produit STRING,
    quantite INT
)
PARTITIONED BY (annee INT, mois INT)
```

```
alter table ventesStatic add partition (annee=2004, mois 1);
alter table ventesStatic add partition (annee=2005, mois 1);
```

12.1 Chargement statique

```
-- Inserer des donnees pour janvier 2004
INSERT INTO TABLE ventesStatic
PARTITION (annee=2004, mois=1)
SELECT id, produit, quantite
FROM ventes
WHERE annee_col=2004 AND mois_col=1;

-- Inserer des donnees pour fevrier 2004
INSERT INTO TABLE ventesStatic
PARTITION (annee=2004, mois=2)
SELECT id, produit, quantite
FROM ventes
WHERE annee_col=2004 AND mois_col=2;
```

13 Bucketing (CLUSTERED BY)

Rappel : Le bucketing répartit les données en fichiers selon un hash d'une ou plusieurs colonnes. Il est utile pour :

- optimiser les **JOINS**,
- accélérer les **SAMPLE**,
- améliorer les performances sur de grands datasets.

```
SET hive.enforce.bucketing = true;

CREATE TABLE clients_bucket (
    id INT,
    nom STRING,
    ville STRING
)
CLUSTERED BY (id)
INTO 8 BUCKETS
STORED AS ORC;
```

14 JOIN optimisé avec des tables bucketées

14.1 Préparation

Deux tables bucketées sur la même colonne et le même nombre de buckets :

```
CREATE TABLE clients (
    id INT,
    nom STRING
)
CLUSTERED BY (id) INTO 8 BUCKETS
STORED AS ORC;
```

```

CREATE TABLE commandes (
    id INT,
    produit STRING,
    montant DOUBLE
)
CLUSTERED BY (id) INTO 8 BUCKETS
STORED AS ORC;

```

Explication clé : Si les deux tables ont :

- la même clé de bucketing,
- le même nombre de buckets,

Hive peut éviter un **shuffle** dans le JOIN.

14.2 JOIN optimisé

```

SELECT c.id, c.nom, o.produit, o.montant
FROM clients c
JOIN commandes o
ON c.id = o.id;

```

Pourquoi est-ce optimisé ? Hive sait que : - le client avec `id = X` et ses commandes sont dans le même bucket, - donc il n'a pas besoin de redistribuer les données en réseau, - le JOIN se fait en lecture locale bucket par bucket.

C'est l'un des principaux avantages du bucketing dans Hive.

15 Conclusion

Ce tutoriel a présenté les commandes fondamentales de Hive avec des explications détaillées et des exemples illustrés. Les sections finales ont introduit les techniques d'optimisation comme le partitionnement et le bucketing, ainsi qu'un exemple concret de JOIN optimisé.