AJC FORMATION

Objectifs

Comprendre la base de données Hive

Découvrir les opportunités de Hive

Acquérir les connaissances nécessaires permettant de mettre en œuvre Hive


Rappel

Rappel

Le terme "**Big Data**" est utilisé pour désigner des collections de grands ensembles de données comprenant un **volume** énorme, une **vitesse** élevée et une **variété** de données qui augmentent de jour en jour.

Rappel

Il est difficile de traiter les Big Data à l'aide des systèmes traditionnels de gestion des données. Par conséquent, **l'Apache Software Foundation** a introduit un cadre appelé **Hadoop** pour résoudre les problèmes de gestion et de traitement des **Big Data**.

Rappel

Hadoop est un framework open-source permettant de stocker et de traiter les Big Data dans un environnement distribué.

MapReduce

Hadoop Distributed File System(HDFS).

Rappel

MapReduce :

C'est un modèle de **programmation parallèle** pour traiter de grandes quantités de données **structurées, semi-structurées et non structurées** sur de grands clusters de matériel de base.

Rappel

HDFS

Hadoop **Distributed File System** est une partie du framework Hadoop, utilisé pour **stocker et traiter** les ensembles de données. Il fournit un système de fichiers **tolérant aux pannes** pour fonctionner sur du matériel de base.

Rappel

L'écosystème Hadoop contient différents **sous-projets (outils)** tels que **Sqoop, Pig et Hive** qui sont utilisés pour **aider les modules Hadoop.**

Sqoop : Il est utilisé pour importer et exporter des données entre HDFS et SGBDR.

Rappel

L'écosystème Hadoop contient différents **sous-projets (outils)** tels que **Sqoop, Pig et Hive** qui sont utilisés pour **aider les modules Hadoop.**

Pig : C'est une plateforme de langage procédural utilisée pour développer un script pour les opérations MapReduce.

Rappel

L'écosystème Hadoop contient différents **sous-projets (outils)** tels que **Sqoop, Pig et Hive** qui sont utilisés pour **aider les modules Hadoop.**

Hive : Il s'agit d'une plateforme utilisée pour développer des scripts de type SQL pour effectuer des opérations MapReduce.

Rappel

Il existe plusieurs façons **d'exécuter des opérations MapReduce**

L'approche traditionnelle utilisant le programme Java MapReduce pour les données structurées, semi-structurées et non structurées.

Rappel

Il existe plusieurs façons **d'exécuter des opérations MapReduce**

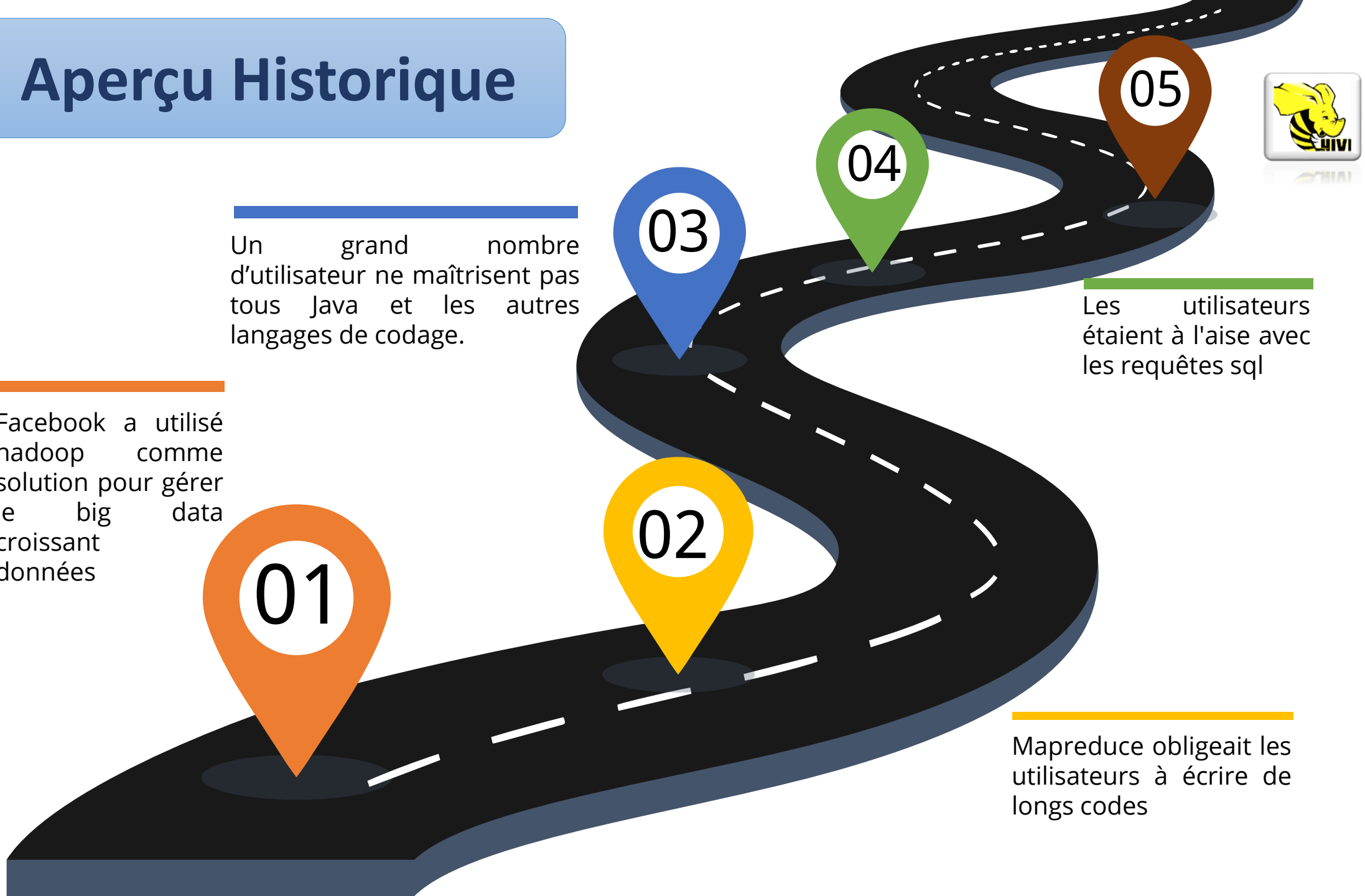
L'approche par script pour MapReduce afin de traiter des données structurées et semi-structurées en utilisant Pig.

Rappel

Il existe plusieurs façons **d'exécuter des opérations MapReduce**

Le langage de requête Hive (HiveQL ou HQL) pour MapReduce afin de traiter les données structurées à l'aide de Hive.

Aperçu Historique



Facebook a utilisé hadoop comme solution pour gérer le big data croissant données

Un grand nombre d'utilisateur ne maîtrisent pas tous Java et les autres langages de codage.

Mapreduce obligeait les utilisateurs à écrire de longs codes

Les utilisateurs étaient à l'aise avec les requêtes sql

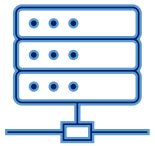
Pourquoi Hive?

Pour le traitement et l'analyse des données, les utilisateurs ont eu des difficultés à coder car ils ne maîtrisaient pas tous les langages de codage.

Les utilisateurs avaient besoin d'un langage similaire au SQL qui était bien connu de tous les utilisateurs

Solution

Hive QL



Preprocessing

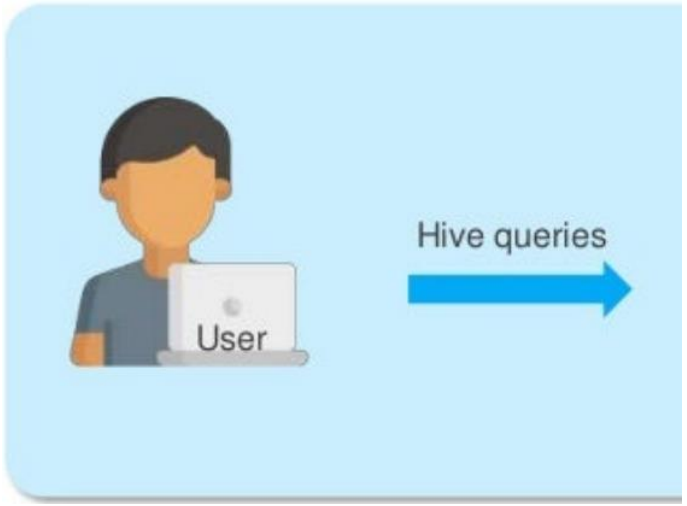
Analyse

Hive, c’est quoi ?

Hive est un système d'entrepôt de données qui est utilisé pour interroger et analyser de grands ensembles de données stockés dans HDFS.

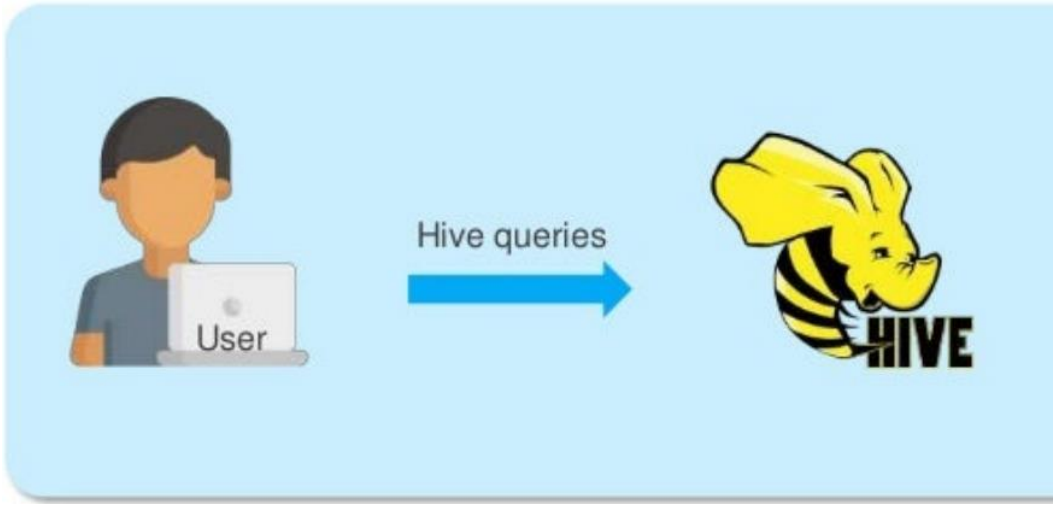
Hive, c’est quoi ?

Hive est un système d'entrepôt de données qui est utilisé pour interroger et analyser de grands ensembles de données stockés dans HDFS.



Hive, c’est quoi ?

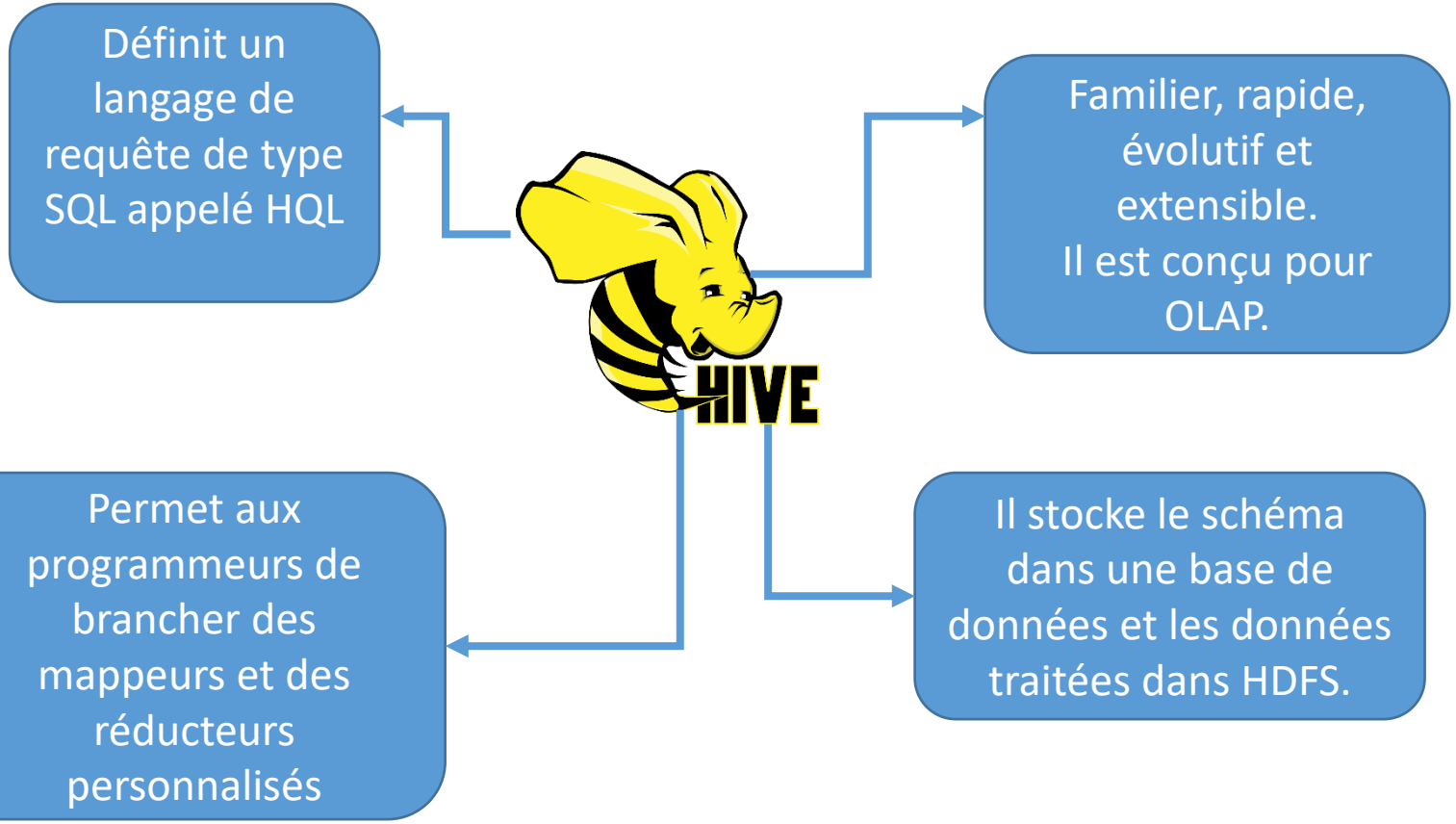
Hive est un système d'entrepôt de données qui est utilisé pour interroger et analyser de grands ensembles de données stockés dans HDFS.



Hive, c’est quoi ?

Hive est un système d'entrepôt de données qui est utilisé pour interroger et analyser de grands ensembles de données stockés dans HDFS.

Principes clés de Hive



Principes clés de Hive

Entrepôt de données

Un système utilisé pour **les rapports et l'analyse des données.** Les DW sont des référentiels centraux de **données intégrées provenant d'une ou de plusieurs sources.**

ETL

processus d'extraction des données **de la source et de les importer dans l'entrepôt de données. Extraire, transformer et charger.**

Motivation

Yahoo a travaillé sur **Pig** pour faciliter le **déploiement** d'applications **sur Hadoop.**

- Leur besoin était principalement axé **sur les données non structurées**

Simultanément, Facebook a commencé à déployer des solutions **d'entrepôt sur Hadoop** qui ont abouti **à Hive.**

- La taille des données en cours de **collecte et d'analyse** dans l'industrie pour **la veille stratégique augmente** rapidement, ce qui rend la solution traditionnelle **extrêmement coûteuse.**

Utilisation de Hive sur Facebook

Hive et Hadoop sont largement utilisés dans Facebook pour différents types d'opérations.

700 To = 2.1 Petabyte après la réplication!

Data model

Hive structure les données en concepts de base de données bien compris tels que:

- **tables, lignes, colonnes, partitions**

Il supporte les types primitifs: **integers, floats, doubles et strings**

Hive prend également en charge:

- **Arrays:** Les éléments du tableau sont de type chaîne de caractères. Les éléments du tableau sont délimités par des virgules. Par exemple, un tableau de fruits est représenté par [apple,banana,orange].

SerDe: une API sérialisée et désérialisée est utilisée pour déplacer des données dans et hors des tables

Query Language (HiveQL)

DDL :

- CREATE DATABASE
- CREATE TABLE
- ALTER TABLE
- SHOW TABLE
- DESCRIBE

DML :

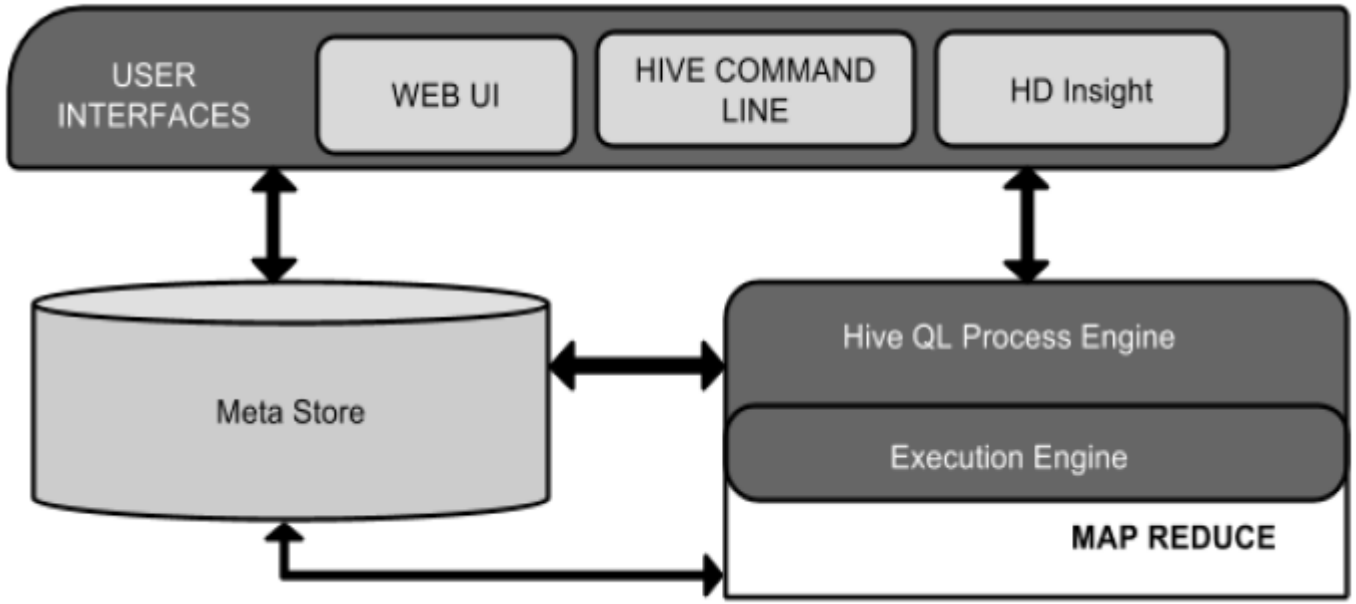
- LOAD TABLE
- INSERT

QUERY :

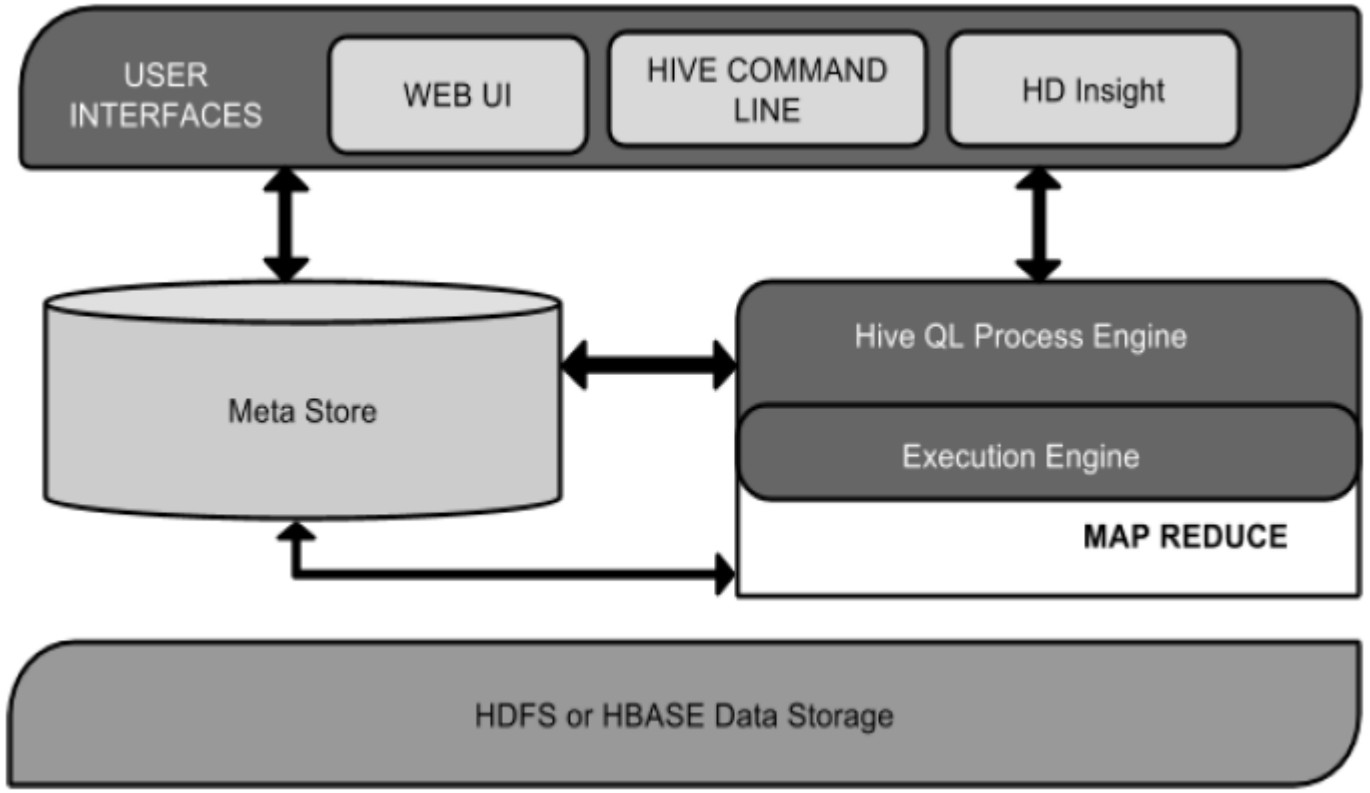
- SELECT
- GROUP BY
- JOIN

Architecture of Hive

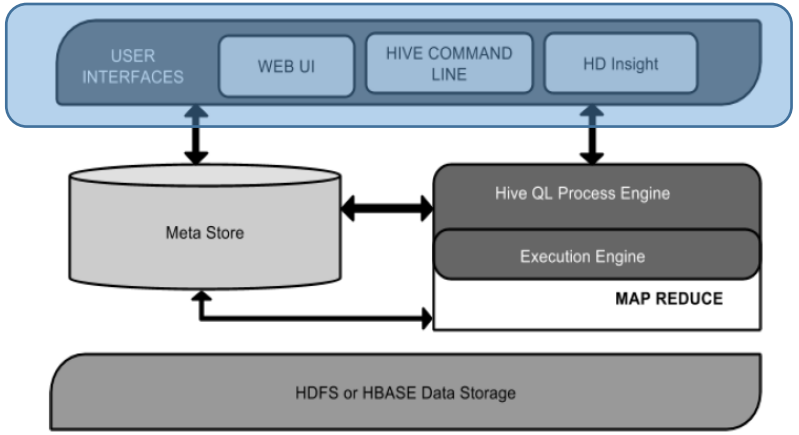
Architecture of Hive



Architecture of Hive



User Interface

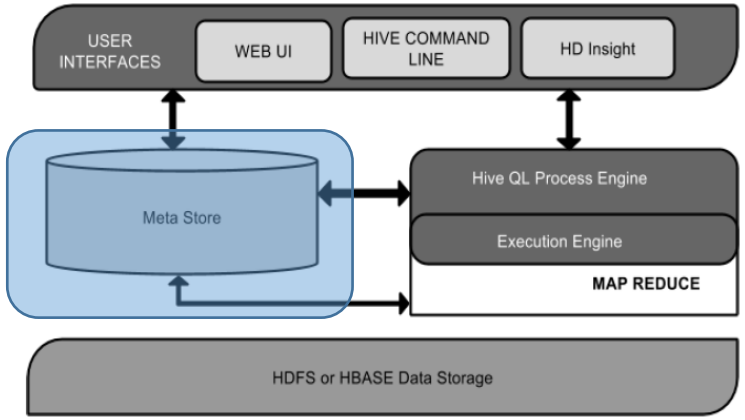


Hive est un logiciel d'infrastructure **d'entrepôt de données** pouvant créer une **interaction entre l'utilisateur et HDFS.**

Les **interfaces utilisateur** prises en charge par Hive :

- Web UI,
- Hive command-line interface (CLI),
- HD Insight (sur un serveur Windows).

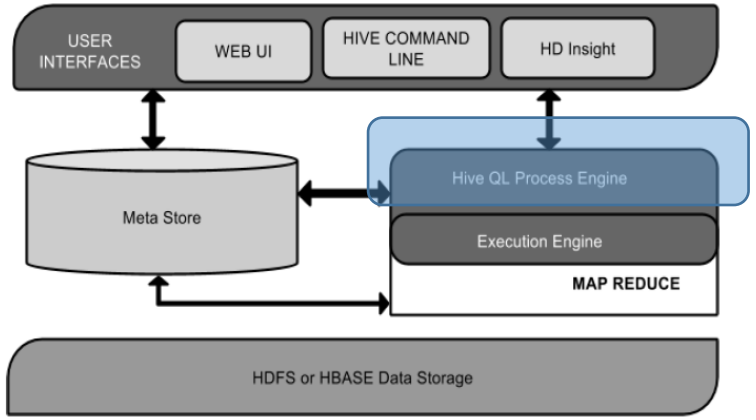
Meta Store



Hive stocke

- le schéma ou les métadonnées des tables,
- bases de données,
- colonnes d'une table,
- types de données et le mappage HDFS.

HiveQL Process Engine

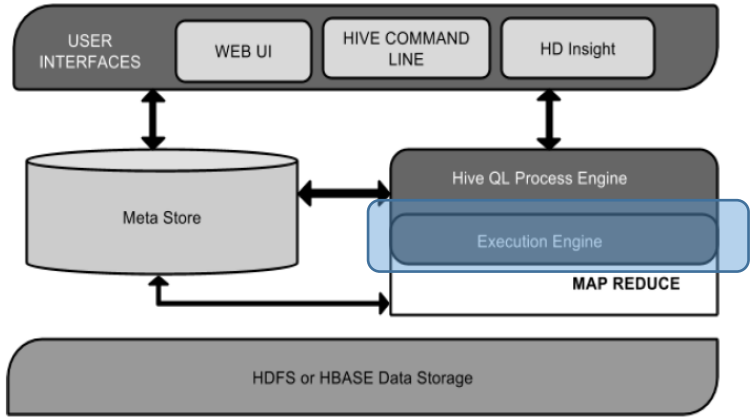


HiveQL est similaire au SQL pour interroger des informations de schéma sur **Metastore.**

C'est l'un des remplacements de **l'approche traditionnelle** du **programme MapReduce.**

Au lieu d'écrire le **programme MapReduce en Java**, nous pouvons écrire une requête pour **un job MapReduce et le traiter.**

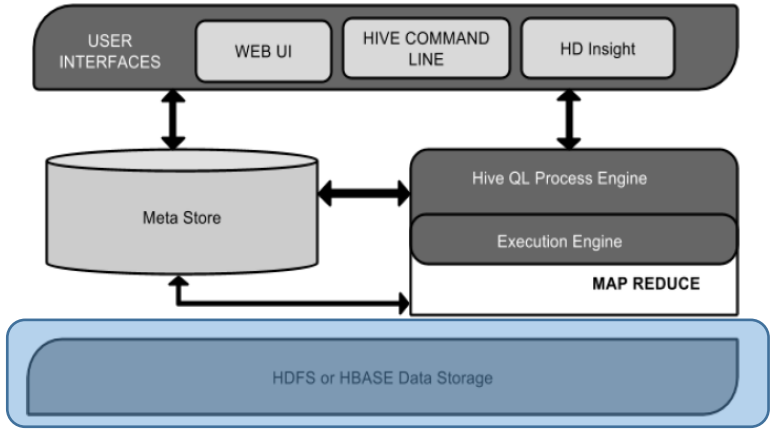
Execution Engine



La partie conjonction de **HiveQL process Engine** et **MapReduce** est **Hive Execution Engine**.

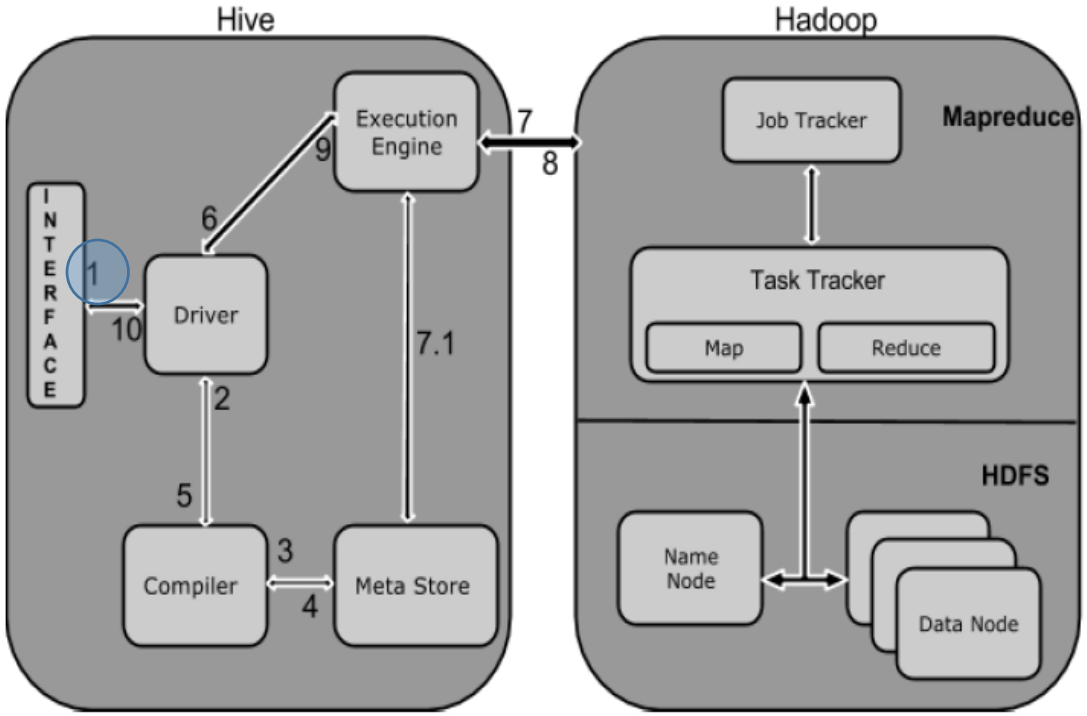
Le moteur d'exécution traite la requête et génère des résultats identiques à ceux de MapReduce.

Execution Engine



Le système de fichiers distribué **Hadoop ou HBASE** sont les techniques de stockage de données permettant de **stocker des données dans un système de fichiers.**

Fonctionnement de Hive

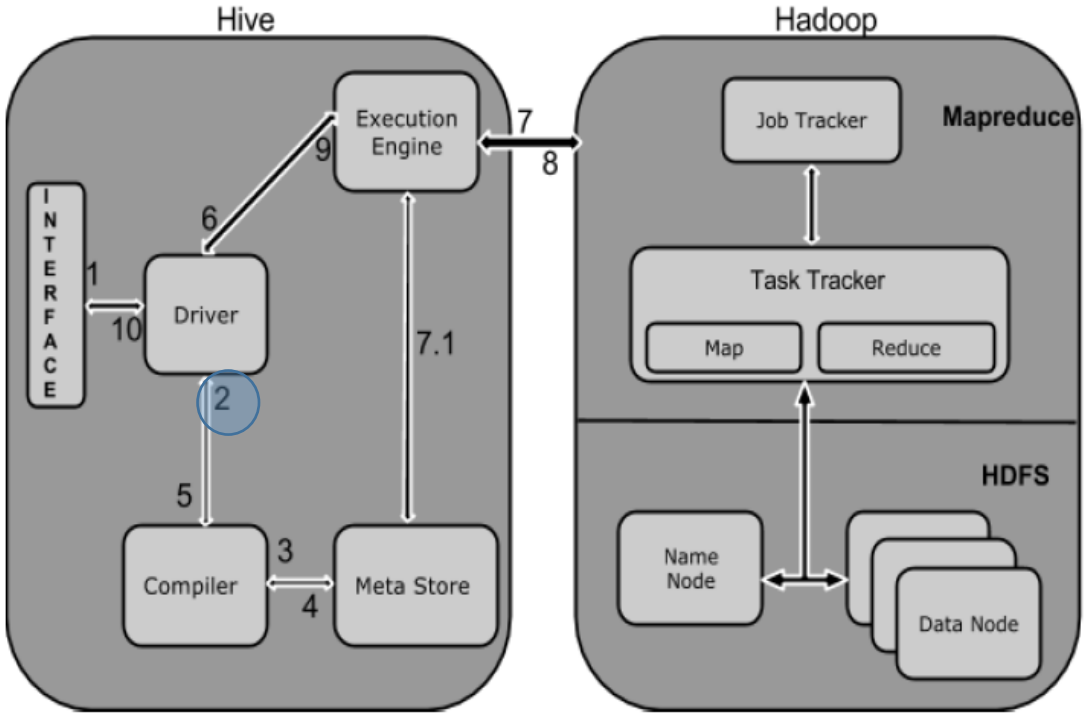


1 - Execute Query

L'interface Hive telle que Ligne de commande ou Web UI envoie une requête au Driver à exécuter.

Driver : tout pilote de base de données tel que JDBC, ODBC, etc.

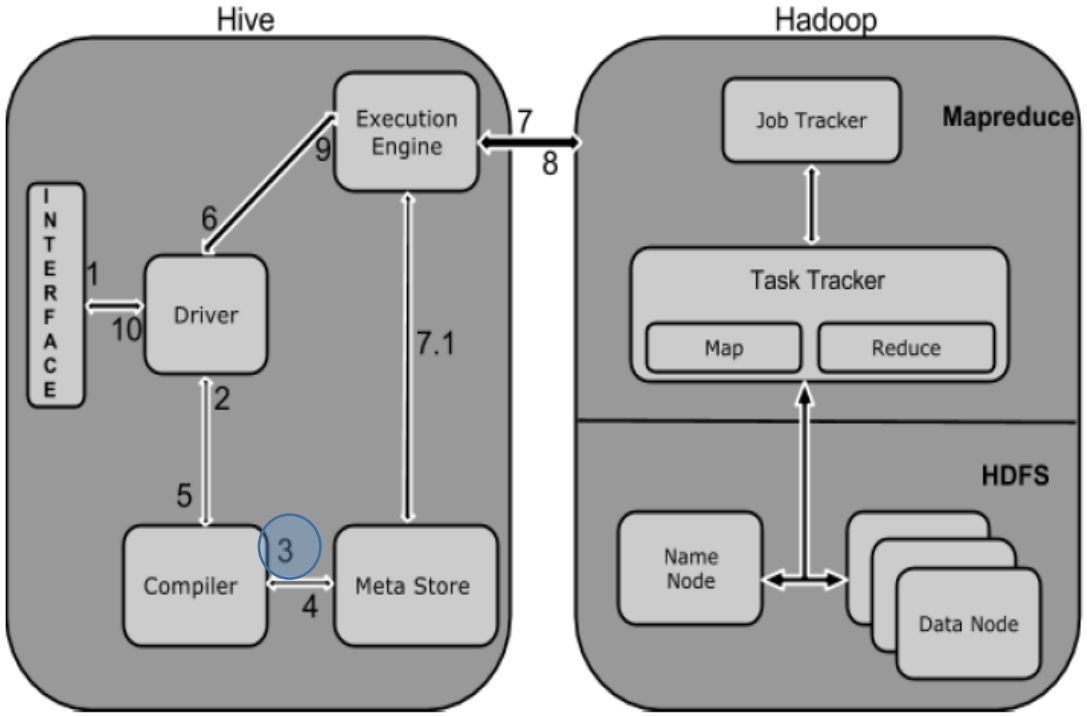
Fonctionnement de Hive



2 - Get Plan (Obtenir un plan)

Le pilote utilise le compilateur de requête qui analyse la requête pour vérifier la syntaxe et le plan de requête ou l'exigence de la requête.

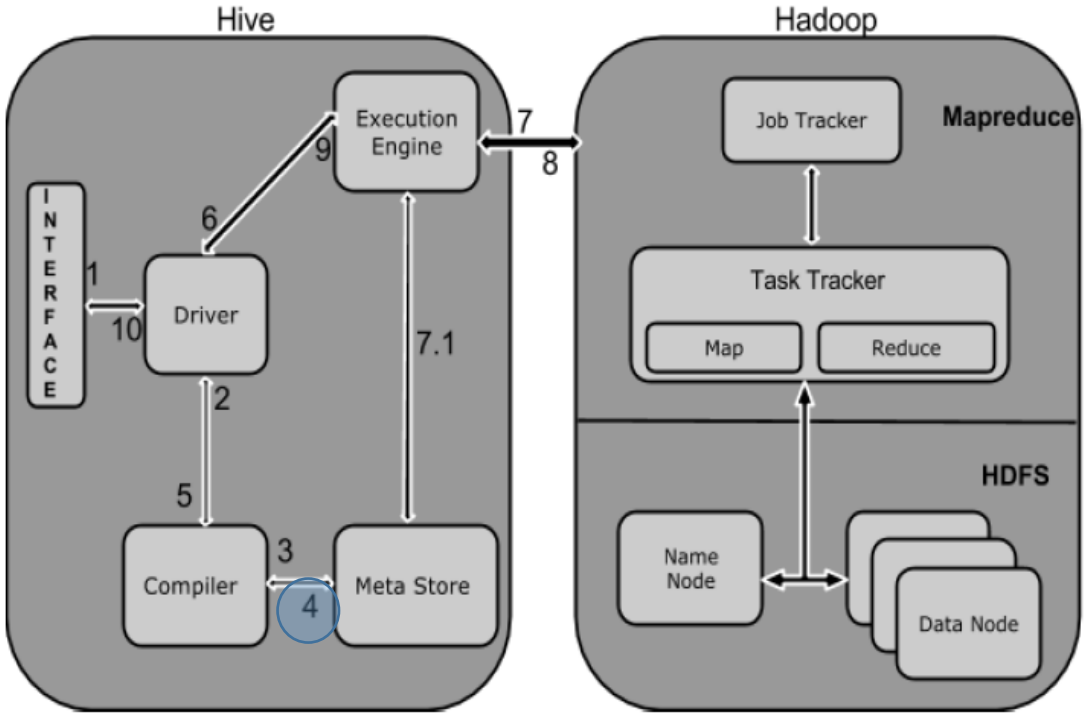
Fonctionnement de Hive



3 - Get Metadata

Le compilateur envoie une requête de métadonnées à Metastore (toute base de données).

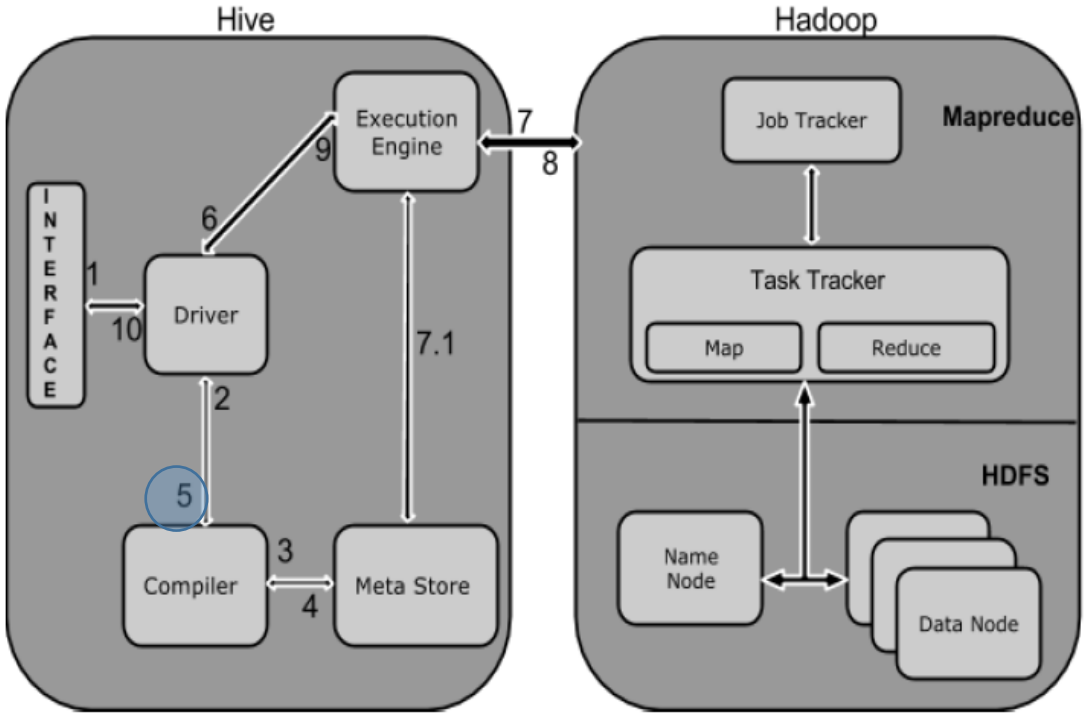
Fonctionnement de Hive



4 – Send Metadata

Metastore envoie des métadonnées en réponse au compilateur.

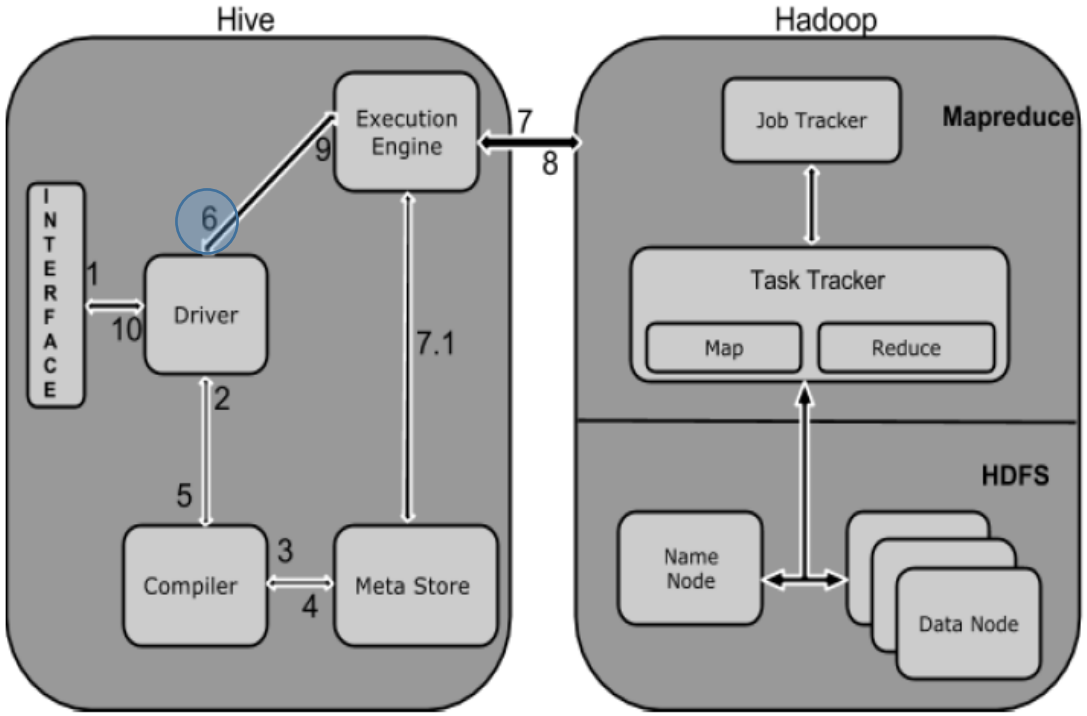
Fonctionnement de Hive



5 – Send Plan

Le compilateur vérifie la configuration requise et renvoie le plan au driver. Jusque là, l'analyse et la compilation d'une requête sont terminées.

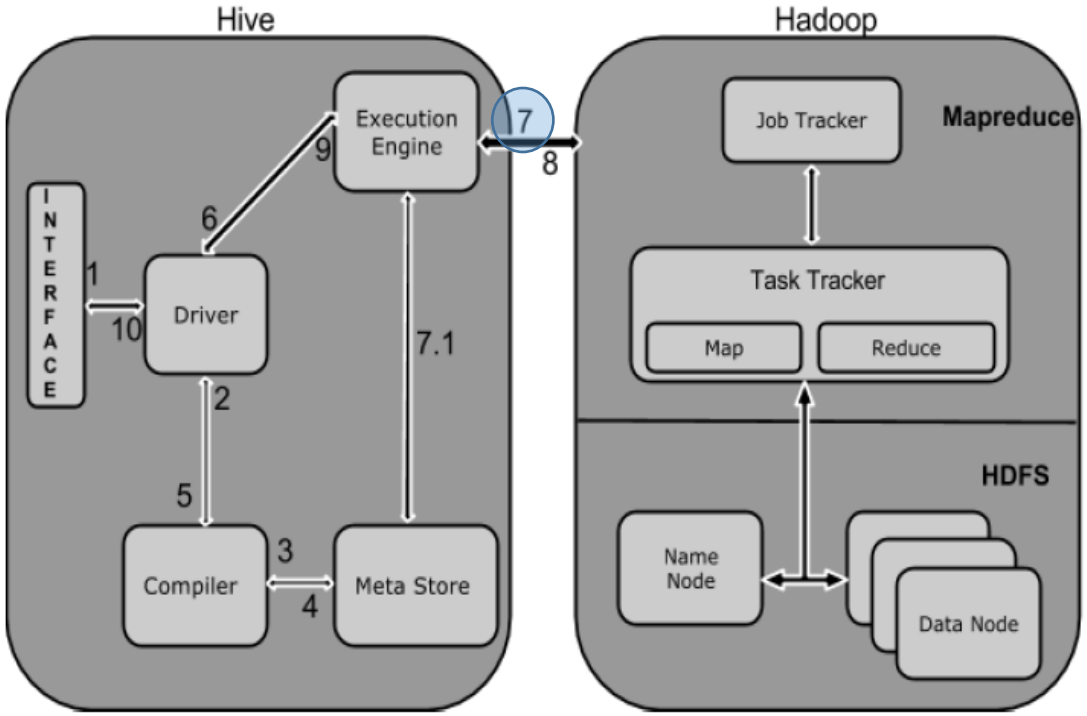
Fonctionnement de Hive



6 – Execute Plan

Le pilote envoie le plan d'exécution au moteur d'exécution

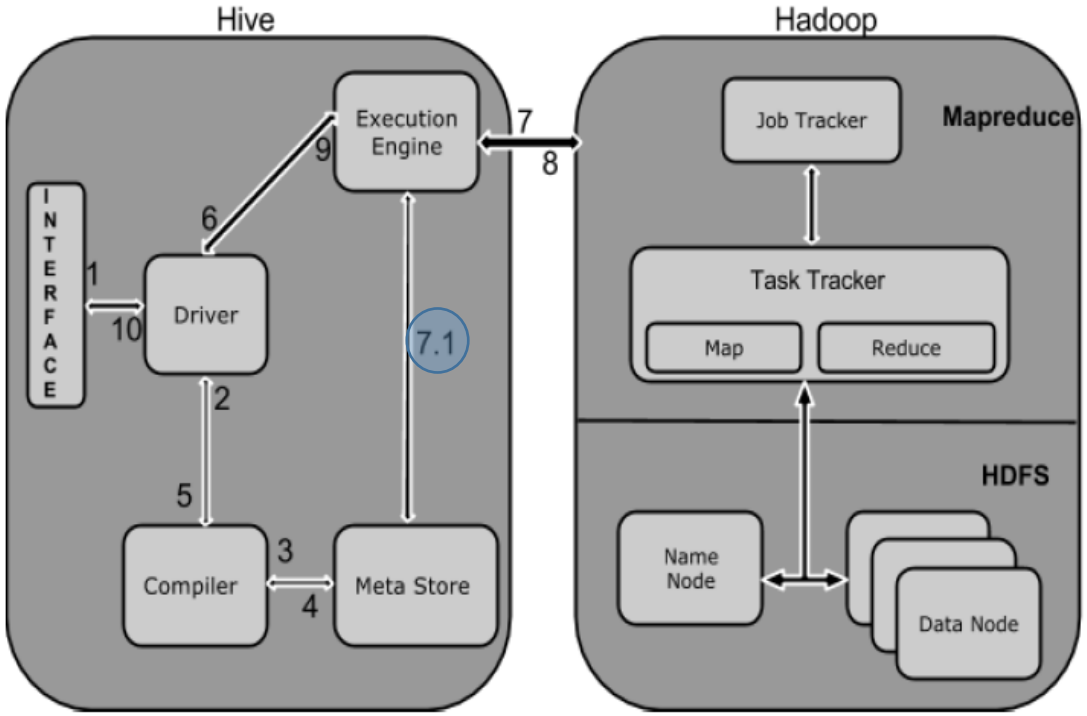
Fonctionnement de Hive



7 – Execute Job

En interne, le processus de l'exécution du job est un **job MapReduce**. Le moteur d'exécution envoie le **job** au **JobTracker**, qui se trouve dans le **NameNode**, et affecte ce job au, **TaskTracker** qui se trouve dans le **DataNode**.
Ici, la requête exécute le **job MapReduce**.

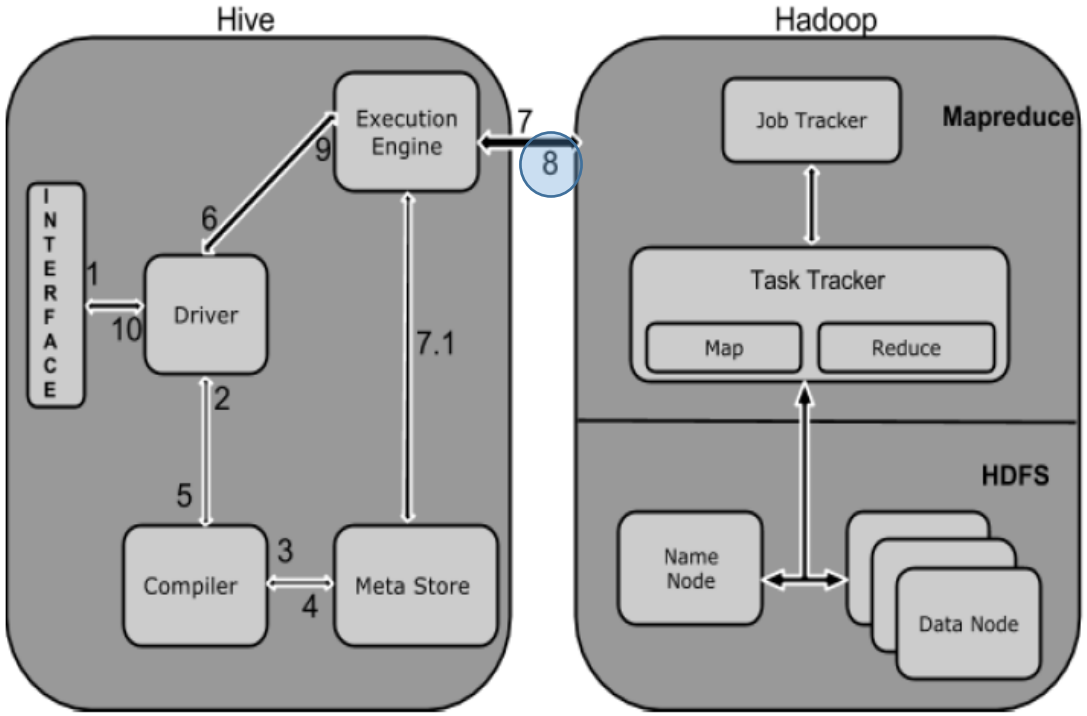
Fonctionnement de Hive



7.1 – Metadata Ops

Pendant l'exécution, le moteur d'exécution peut exécuter des opérations de métadonnées avec Metastore.

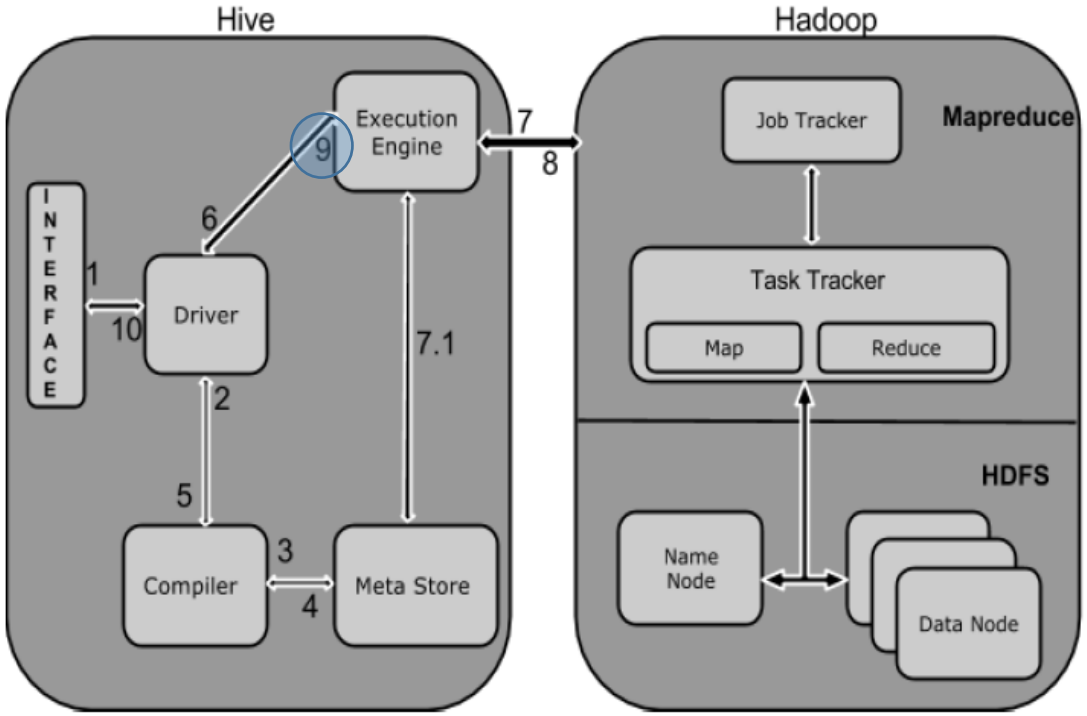
Fonctionnement de Hive



8 – Fetch Result (Récupérer le résultat)

Le moteur d'exécution reçoit les résultats des DataNodes.

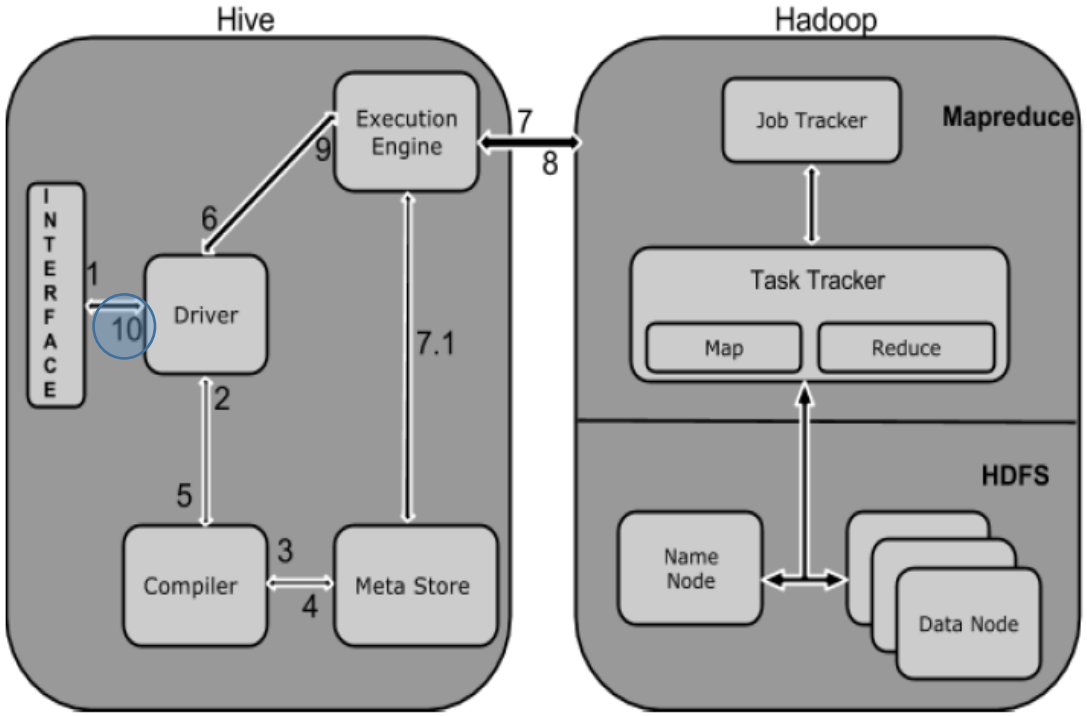
Fonctionnement de Hive



9 – Send Result

Le moteur d'exécution envoie les valeurs résultantes au driver.

Fonctionnement de Hive



10 – Send Result

Le driver envoie les résultats à l'interface Hive.

Hive QL – Join

page_view

pageid	userid	time
1	111	9:08:01
2	111	9:08:13
1	222	9:08:14

x

user

userid	age	gender
111	25	female
222	32	male

=

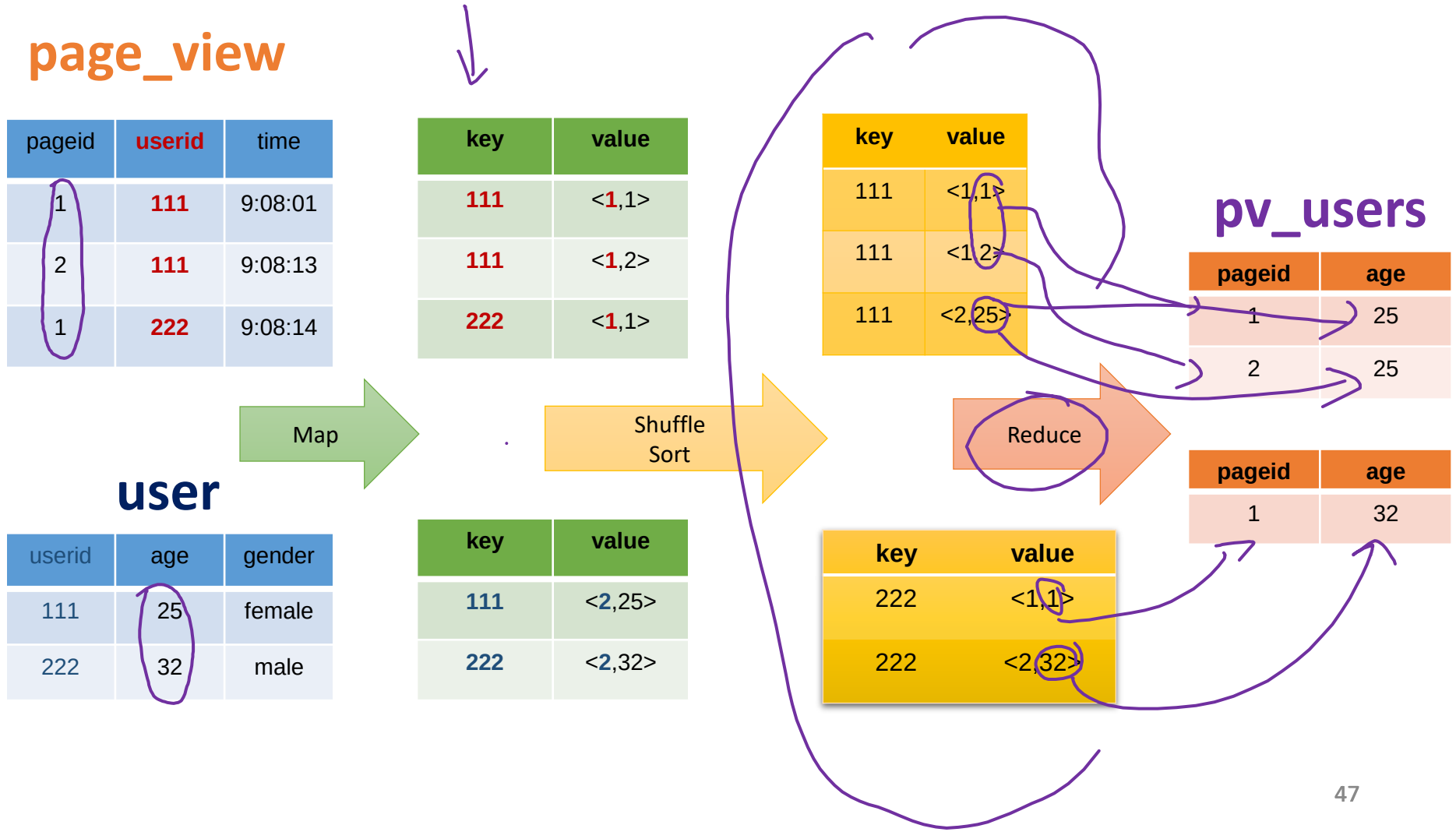
pv_users

pageid	age
1	25
2	25
1	32

- SQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

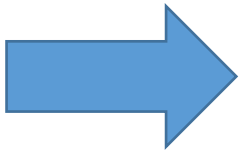
Hive QL – Join MapReduce



Hive QL – Group By

pv_users

pageid	age
1	25
2	25
1	32
2	25



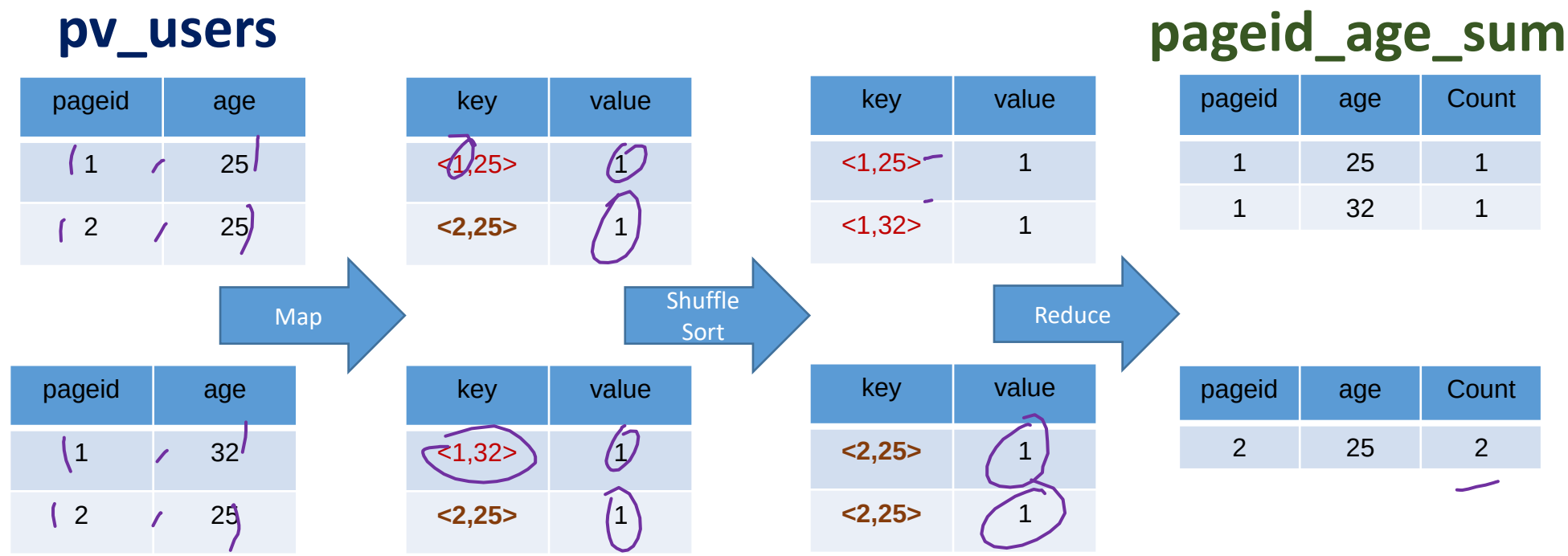
pageid_age_sum

pageid	age	Count
1	25	1
2	25	2
1	32	1

SQL:

```
INSERT INTO TABLE pageid_age_sum
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

Hive QL – Group By in Map Reduce



Hive QL – Group By with Distinct

page_view

pageid	userid	time	pageid
1	111	9:08:01	1
2	111	9:08:13	2
1	222	9:08:14	1
2	111	9:08:20	2

Map

result

pageid	count_distinct_userid
1	2
2	1

SQL

```
SELECT pageid, COUNT(DISTINCT userid)
FROM page_view
GROUP BY pageid
```

Hive QL – Group By with Distinct in Map Reduce

page_view

pageid	userid	time
1	111	9:08:01
2	111	9:08:13
1	222	9:08:14
2	111	9:08:20

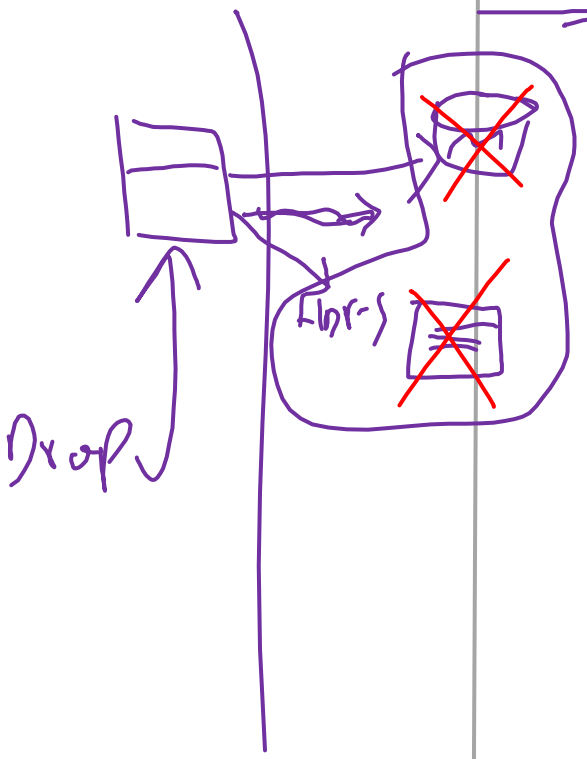
Shuffle Sort

key	v
<1,111>	
<1,222>	
<2,111>	
<2,111>	

Reduce

pageid	count
1	2
2	1

Hive QL – Internal Tables



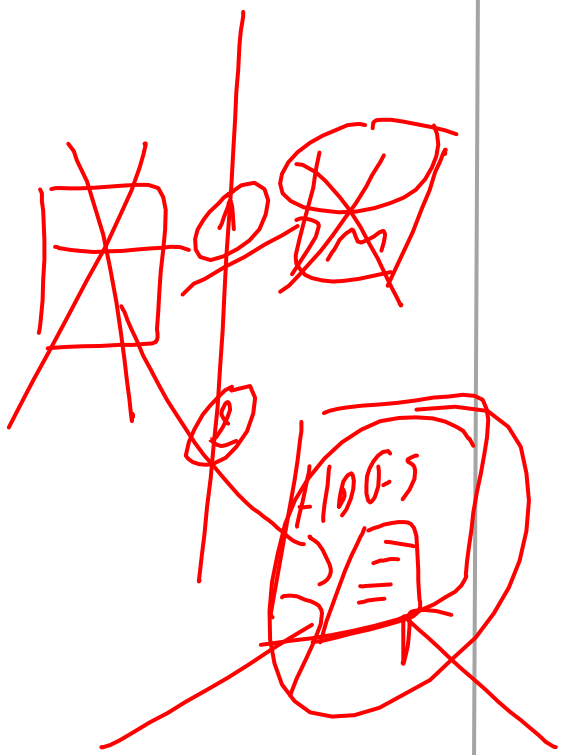
Stockées dans un répertoire basé sur les paramètres de **hive.metastore.warehouse.dir**
les tables internes sont par défaut stockées dans le répertoire suivant **/user/hive/warehouse** (modifiable dans le fichier de configuration).

La suppression de la table supprime les **métadonnées** et les **données** de **master-node et HDFS respectivement**.

La sécurité des fichiers des tables internes est contrôlée uniquement via HIVE.

La sécurité doit être gérée dans HIVE, probablement au niveau du schéma (dépend de l'organisation).

Hive QL – External Tables



Une table externe **stocke** des **fichiers** sur le serveur **HDFS**, mais **les tables ne sont pas complètement liées au fichier source**.

Si vous **supprimez** une **table externe**, le fichier **reste** sur le serveur **HDFS**.

Par exemple, si vous créez une table externe appelée «**table_test**» dans HIVE, à l'aide de HIVE-QL et liez la table au fichier «**fichier**», la **suppression** de «**table_test**» de HIVE ne supprimera pas «**fichier**» de **HDFS**.

Les **fichiers** de table externes sont accessibles à toute personne ayant accès à la structure de fichier **HDFS**. Par conséquent, la sécurité doit être gérée au niveau du fichier/dossier HDFS.

Les **métadonnées** sont conservées sur le nœud maître et la **suppression** d'une **table externe** de HIVE **supprime** uniquement les **métadonnées** et non les données/fichiers.

DDL : Data Definition Language

CREATE DATABASE

Hive est une technologie de base de données qui permet **de définir des bases de données** et **des tables** pour **analyser des données structurées**. Le thème de l'analyse des données structurées consiste à stocker les données **sous forme des tables**, et à passer des requêtes **pour les analyser.** Hive contient **une base de données** par défaut nommée *default*.

DDL : Data Definition Language

CREATE DATABASE

Une instruction utilisée pour créer une base de données dans Hive.
Une base de données dans Hive est **un namespace ou une collection** de tables.

DDL: Data Definition Language

- CREATE DATABASE

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>;
```

IF NOT EXISTS est une clause facultative, qui notifie à l'utilisateur qu'une base de données portant le même nom existe déjà. Nous pouvons utiliser SCHEMA à la place de DATABASE dans cette commande.

Exemple

```
CREATE DATABASE [IF NOT EXISTS] userdata;
CREATE SCHEMA userdata;
```

DDL: Data Definition Language

La requête suivante est utilisée pour vérifier une liste de bases de données :

```
hive> SHOW DATABASES;
default
userdb
```

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveCreateDb {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

   public static void main(String[] args) throws SQLException {
      // Register driver and create driver instance

      Class.forName(driverName);
      // get connection

      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("CREATE DATABASE userdb");
      System.out.println("Database userdb created successfully.");
      con.close();
   }
}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveCreateDb.java. Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveCreateDb.java
$ java HiveCreateDb
```

Create userdb database successful.

DDL: Data Definition Language

Drop Database est une instruction qui supprime toutes les tables et la base de données. Sa syntaxe est la suivante :

```
DROP DATABASE (DATABASE|SCHEMA) [IF EXISTS]
database_name
```

Les requêtes suivantes sont utilisées pour supprimer une base de données. Supposons que le nom de la base de données est userdb

```
DROP DATABASE IF EXISTS userdb;
```

DDL: Data Definition Language

Drop Database est une instruction qui supprime toutes les tables et la base de données. Sa syntaxe est la suivante :

```
DROP DATABASE (DATABASE|SCHEMA) [IF EXISTS]
database_name
```

La requête suivante supprime la base de données en utilisant CASCADE. Cela signifie qu'il faut supprimer les tables respectives avant de supprimer la base de données.

```
DROP DATABASE IF EXISTS userdb CASCADE;
```

DDL: Data Definition Language

Drop Database est une instruction qui supprime toutes les tables et la base de données. Sa syntaxe est la suivante :

```
DROP DATABASE (DATABASE|SCHEMA) [IF EXISTS]
database_name
```

La requête suivante vide la base de données en utilisant SCHEMA.

```
DROP SCHEMA userdb;
```

Cette clause a été ajoutée dans Hive 0.6.

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveDropDb {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {

      // Register driver and create driver instance
      Class.forName(driverName);

      // get connection
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("DROP DATABASE userdb");
      System.out.println("Drop userdb database successful.");
      con.close();
   }
}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveDropDb.java. Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveDropDb.java
$ java HiveDropDb
```

Drop userdb database successful.

DDL: Data Definition Language

Create Table est une instruction utilisée pour créer une table dans Hive. La syntaxe et l'exemple sont les suivants:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT
EXISTS] [db_name.] table_name
[(col_name data_type [COMMENT col_comment],
...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

DDL: Data Definition Language

Supposons que vous deviez créer **une table nommée employee** à l'aide de l'instruction CREATE TABLE. Le tableau suivant présente les champs et leurs types de données dans la table employee:

Sr.No	Field Name	Data Type
1	Eid	int
2	Name	String
3	Salary	Float
4	Designation	string

DDL: Data Definition Language

Les données suivantes sont un commentaire, des champs formatés en ligne tels que le terminateur de champ, le terminateur de ligne et le type de fichier stocké.

```
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

DDL: Data Definition Language

La requête complète :

```
CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Si vous ajoutez l'option IF NOT EXISTS, Hive ignore l'instruction au cas où la table existe déjà.

Lorsque la création de la table est réussie, vous obtenez la réponse suivante

OK
Time taken: 5.905 seconds

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveCreateTable {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
        Statement stmt = con.createStatement();
        stmt.executeQuery("CREATE TABLE IF NOT EXISTS "
         +" employee ( eid int, name String, "
         +" salary String, destignation String)"
         +" COMMENT 'Employee details'"
         +" ROW FORMAT DELIMITED"
         +" FIELDS TERMINATED BY '\t'"
         +" LINES TERMINATED BY '\n'"
         +" STORED AS TEXTFILE;");
      System.out.println(" Table employee created.");
      con.close();}}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveCreateTable.java. Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveCreateDb.java
$ java HiveCreateDb
```

Table employee created.

DDL: Data Definition Language

ALTER TABLE est utilisé pour modifier une table dans Hive.

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

Exemple :

```
ALTER TABLE clients RENAME TO clts;
```

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveAlterRenameTo {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("ALTER TABLE clients RENAME TO clts;");
      System.out.println("Table Renamed Successfully");
      con.close();
   }
}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveAlterRenameTo.java
Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveAlterRenameTo.java
$ java HiveAlterRenameTo
```

Table renamed successfully.

DDL: Data Definition Language

Le tableau suivant contient les champs du tableau des employés et indique les champs à modifier (en gras).

Field Name	Convert from Data Type	Change Field Name	Convert to Data Type
eid	int	eid	int
name	String	**ename**	String
salary	**Float**	salary	**Double**
designation	String	designation	String

```
ALTER TABLE employee CHANGE name ename String;
ALTER TABLE employee CHANGE salary salary Double;
```

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveAlterChangeColumn {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      // Register driver and create driver instance
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("ALTER TABLE employee CHANGE name ename String;");
      stmt.executeQuery("ALTER TABLE employee CHANGE salary salary Double;");
      System.out.println("Change column successful.");
      con.close();
   }
}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveAlterChangeColumn.java.
Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveAlterChangeColumn.java
$ java HiveAlterChangeColumn
```

Change column successfully.

DDL: Data Definition Language

Ajouter une colonne « Dept » à la table employee

```
ALTER TABLE employee ADD COLUMNS (
dept STRING COMMENT 'Department name');
```

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveAlterChangeColumn {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      // Register driver and create driver instance
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("ALTER TABLE employee ADD COLUMNS " + " (dept STRING
COMMENT 'Department name');");

      System.out.println("Add column successful.");
      con.close();
   }
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nomméHiveAlterAddColumn.java.
Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveAlterAddColumn.java
$ java HiveAlterAddColumn
```

Add column successfully.

DDL: Data Definition Language

La requête suivante supprime toutes les colonnes de la table employee et les remplace par les colonnes empid et name :

```
ALTER TABLE employee REPLACE COLUMNS (
eid INT empid Int,
ename STRING name String);
```

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveAlterChangeColumn {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      // Register driver and create driver instance
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("ALTER TABLE employee REPLACE COLUMNS "
         +" (eid INT empid Int,"
         +" ename STRING name String);");

      System.out.println("Replace column successful.");
      con.close();
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveAlterReplaceColumn.java.
Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveAlterReplaceColumn.java
$ java HiveAlterReplaceColumn
```

Replace column successfully.

DDL: Data Definition Language

DROP TABLE : Pour supprimer une table.

```
DROP TABLE [IF EXISTS] table_name;
```

La requête suivante supprime une table nommée employee :

```
DROP TABLE IF EXISTS employee;
```

OK
Time taken: 5.3 seconds

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveDropTable {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
       Class.forName(driverName);
       Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();

      stmt.executeQuery("DROP TABLE IF EXISTS employee;");
      System.out.println("Drop table successful.");

      con.close();
   }
}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveDropTable.java.
Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveDropTable.java
$ java HiveDropTable
```

Drop table successfully.

DDL: Data Definition Language

La requête suivante est utilisée pour vérifier la liste des tables :

```
hive> SHOW TABLES;
emp
ok
Time taken: 2.1 seconds
hive>
```

DDL: Data Definition Language

Ajouter une colonne « Dept » à la table employee

```
ALTER TABLE employee ADD COLUMNS (
dept STRING COMMENT 'Department name');
```

DDL

- ✓ SHOW

```
SHOW (DATABASES|SCHEMAS) [LIKE 'identifier_with_wildcards'];
SHOW TABLES [IN database_name] ['identifier_with_wildcards'];
SHOW VIEWS [IN/FROM database_name] [LIKE
'pattern_with_wildcards'];
SHOW PARTITIONS table_name;
```

Exemple :

```
SHOW DATABASES;
SHOW SCHEMAS;
SHOW TABLES;
SHOW VIEWS;
SHOW PARTITIONS table_name PARTITION(ds='2010-03-03');
```

DDL

- ✓ DESCRIBE

```
DESCRIBE DATABASE [EXTENDED] db_name;
DESCRIBE SCHEMA [EXTENDED] db_name;     -- (Note: Hive 1.1.0
and later)
DESCRIBE Table/View/Column
```

Exemple

```
DESCRIBE userdata;
DESCRIBE clients;
```

DML: Data Manipulation Language

LOAD DATA

Généralement, après avoir créé **une table en SQL**, nous pouvons insérer des données en utilisant **l'instruction Insert**. Mais dans Hive, nous pouvons insérer des données en utilisant **l'instruction LOAD DATA.**

Lors de l'insertion de données dans Hive, il est préférable d'utiliser **LOAD DATA** pour **stocker des enregistrements en masse**. Il y a deux façons de charger des données : la première est à partir du système de fichiers local et la seconde à partir du système de fichiers Hadoop.

DML: Data Manipulation Language

La syntaxe LOAD DATA

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE
tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]
```

LOCAL est un identifiant permettant de spécifier le chemin local. Il est facultatif.
OVERWRITE est facultatif pour écraser les données dans la table.
PARTITION est facultatif.

Exemple :

```
CREATE TABLE tab1 (col1 int, col2 int)
PARTITIONED BY (col3 int) STORED AS ORC;
LOAD DATA LOCAL INPATH 'filepath' INTO
TABLE tab1;
```

DML: Data Manipulation Language

Nous allons insérer les données suivantes dans la table. Il s'agit d'un fichier texte nommé sample.txt dans le répertoire /home/user.

```
1201 Gopal 45000 Technical manager
1202 Manisha 45000 Proof reader
1203 Masthanvali 40000 Technical writer
1204 Kiran 40000 Hr Admin
1205 Kranthi 30000 Op Admin
```

La requête suivante charge le texte donné dans la table.

```
LOAD DATA LOCAL INPATH '/home/user/sample.txt'
OVERWRITE INTO TABLE employee;
```

OK
Time taken: 15.905 seconds

DDL: Data Definition Language

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveLoadData {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      stmt.executeQuery("LOAD DATA LOCAL INPATH '/home/user/sample.txt'" +
"OVERWRITE INTO TABLE employee;");
      System.out.println("Load Data into employee successful");

      con.close();}}
```

DDL: Data Definition Language

Enregistrez le programme dans un fichier nommé HiveLoadTable.java. Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveLoadData.java
$ java HiveLoadData
```

Load Data into employee successful

DML

- ✓ INSERT

```
Standard Syntax:
INSERT INTO TABLE tablename [PARTITION (partcol1[=val1],
partcol2[=val2] ...)] VALUES values_row [, values_row ...]

Where values_row is:
( value [, value ...] )
where a value is either null or any valid SQL literal
```

DML

- ✓ INSERT Exemple

```
CREATE TABLE students (name VARCHAR(64), age INT, gpa
DECIMAL(3, 2))
  CLUSTERED BY (age) INTO 2 BUCKETS STORED AS ORC;

INSERT INTO TABLE students
  VALUES ('fred flintstone', 35, 1.28), ('barney rubble', 32,
2.32);
```

DML

- ✓ INSERT

```
Standard syntax:
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select_statement1 FROM from_statement;
INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1 FROM from_statement;

Hive extension (multiple inserts):
FROM from_statement
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select_statement1
[INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select_statement2]
[INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2] ...;
FROM from_statement
INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1
[INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2]
[INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select_statement2] ...;

Hive extension (dynamic partition inserts):
INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select_statement FROM from_statement;
INSERT INTO TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select_statement FROM from_statement;
```

DML

- ✓ INSERT Exemple

```
FROM page_view_stg pvs
INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-
08', country)
SELECT pvs.viewTime, pvs.userid, pvs.page_url,
pvs.referrer_url, null, null, pvs.ip, pvs.cnt;
```

Partitions

Hive organise les tables en partitions. C'est une façon de diviser une table en parties connexes basées sur les valeurs des colonnes partitionnées telles que **la date**, **la ville et le département**. En utilisant la partition, il est facile d'interroger une partie des données.

Les tables ou partitions sont subdivisées en plusieurs parties (buckets), afin de fournir une structure supplémentaire aux données qui peuvent être utilisées pour une interrogation plus efficace.

Partitions

Le Bucketing fonctionne sur la base de la valeur de la fonction de hachage de certaines colonnes d'une table.

Par exemple, une table nommée Tab1 contient des données sur les employés telles que l'identifiant, le nom, le département et l'année d'entrée en fonction.

Supposons que vous ayez besoin de récupérer les détails de tous les employés qui ont été embauchés en **2012.** Une requête recherche les informations requises dans l'ensemble de la table. Cependant, si vous partitionnez les données des employés en fonction de l'année et que vous les stockez dans un fichier séparé, vous réduisez le temps de traitement de la requête.

Partitions

L'exemple suivant montre comment partitionner un fichier et ses données :Le fichier suivant contient la table employeedata.

/tab1/employeedata/file1

```
id, name, dept, yoj
1, gopal, TP, 2012
2, kiran, HR, 2012
3, kaleel,SC, 2013
4, Prasanth, SC, 2013
```

Partitions

Les données sont partitionnées en deux fichiers:

/tab1/employeedata/2012/file2

```
1, gopal, TP, 2012
2, kiran, HR, 2012
```

/tab1/employeedata/2013/file3

```
3, kaleel,SC, 2013
4, Prasanth, SC, 2013
```

Partitions

Nous pouvons ajouter des partitions à une table en modifiant la table. Supposons que nous ayons une table appelée employee avec des champs tels que Id, Name, Salary, Designation, Dept, et yoj.

```
ALTER TABLE table_name ADD [IF NOT
EXISTS] PARTITION partition_spec
[LOCATION 'location1'] partition_spec
[LOCATION 'location2'] ...;

partition_spec:
: (p_column = p_col_value, p_column =
p_col_value, ...)
```

Partitions

La requête suivante est utilisée pour ajouter une partition à la table des employés.

```
hive> ALTER TABLE employee
> ADD PARTITION (year='2012')
> location '/2012/part2012';
```

Partitions

La requête suivante est utilisée pour renommer une partition:

```
ALTER TABLE table_name PARTITION
partition_spec RENAME TO PARTITION
partition_spec2;
```

Exemple

```
ALTER TABLE employee PARTITION
(year='1203')RENAME TO PARTITION
(Yoj='1203');
```

Partitions

La requête suivante est utilisée pour supprimer une partition:

```
ALTER TABLE table_name DROP [IF EXISTS]
PARTITION partition_spec, PARTITION
partition_spec,...;
```

Exemple

```
ALTER TABLE employee DROP [IF EXISTS]
PARTITION (year='1203');
```

Opérateurs intégrés de Hive

Il existe quatre types d'opérateurs dans Hive :

- Les opérateurs relationnels
- Opérateurs arithmétiques
- Opérateurs logiques
- Opérateurs complexes

Opérateurs intégrés de Hive

Les opérateurs relationnels: Ces opérateurs sont utilisés pour comparer deux opérandes. Le tableau suivant décrit les opérateurs relationnels disponibles dans Hive:

Operator	Operand	Description
A = B	all primitive types	TRUE if expression A is equivalent to expression B otherwise FALSE.
A != B	all primitive types	TRUE if expression A is not equivalent to expression B otherwise FALSE.
A < B	all primitive types	TRUE if expression A is less than expression B otherwise FALSE.
A <= B	all primitive types	TRUE if expression A is less than or equal to expression B otherwise FALSE.
A > B	all primitive types	TRUE if expression A is greater than expression B otherwise FALSE.
A >= B	all primitive types	TRUE if expression A is greater than or equal to expression B otherwise FALSE.
A IS NULL	all types	TRUE if expression A evaluates to NULL otherwise FALSE.
A IS NOT NULL	all types	FALSE if expression A evaluates to NULL otherwise TRUE.
A LIKE B	Strings	TRUE if string pattern A matches to B otherwise FALSE.
A RLIKE B	Strings	NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B , otherwise FALSE.
A REGEXP B	Strings	Same as RLIKE.

Opérateurs intégrés de Hive

Les opérateurs relationnels: Exemple

Supposons que la table des employés est composée de champs nommés Id, Nom, Salaire, Désignation, et Département comme indiqué ci-dessous. Générez une requête pour récupérer les détails de l'employé dont l'Id est 1205.

```
+-----+--------------+--------+---------------------------+------+
| Id  | Name         | Salary | Designation               | Dept |
+-----+--------------+--------+---------------------------+------+
|1201 | Gopal        | 45000  | Technical manager         | TP   |
|1202 | Manisha      | 45000  | Proofreader               | PR   |
|1203 | Masthanvali  | 40000  | Technical writer          | TP   |
|1204 | Krian        | 40000  | Hr Admin                  | HR   |
|1205 | Kranthi      | 30000  | Op Admin                  | Admin|
+-----+--------------+--------+---------------------------+------+
```

Opérateurs intégrés de Hive

Les opérateurs relationnels: Exemple

La requête suivante est exécutée pour récupérer les détails de l'employé en utilisant la table ci-dessus :

```
SELECT * FROM employee WHERE Id=1205;
```

```
+-----+-----------+-----------+----------------------------------+
| ID  | Name      | Salary    | Designation                      | Dept  |
+-----+---------------+-------+----------------------------------+
|1205 | Kranthi   | 30000     | Op Admin                         | Admin |
+-----+-----------+-----------+----------------------------------+
```

Opérateurs intégrés de Hive

Les opérateurs relationnels: Exemple

La requête suivante est exécutée pour récupérer les détails de l'employé dont le salaire est supérieur ou égal à Rs 40000

```
SELECT * FROM employee WHERE Salary>=40000;
```

```
+-----+------------+--------+----------------------------+------+
| ID  | Name       | Salary | Designation                | Dept |
+-----+------------+--------+----------------------------+------+
|1201 | Gopal      | 45000  | Technical manager          | TP   |
|1202 | Manisha    | 45000  | Proofreader                | PR   |
|1203 | Masthanvali| 40000  | Technical writer           | TP   |
|1204 | Krian      | 40000  | Hr Admin                   | HR   |
+-----+------------+--------+----------------------------+------+
```

Opérateurs intégrés de Hive

Opérateurs arithmétiques : Ces opérateurs prennent en charge diverses opérations arithmétiques courantes sur les opérandes. Ils retournent tous des types de nombres. Le tableau suivant décrit les opérateurs arithmétiques disponibles dans Hive :

Operators	Operand	Description
A + B	all number types	Gives the result of adding A and B.
A - B	all number types	Gives the result of subtracting B from A.
A * B	all number types	Gives the result of multiplying A and B.
A / B	all number types	Gives the result of dividing B from A.
A % B	all number types	Gives the reminder resulting from dividing A by B.
A & B	all number types	Gives the result of bitwise AND of A and B.
A | B	all number types	Gives the result of bitwise OR of A and B.
A ^ B	all number types	Gives the result of bitwise XOR of A and B.
~A	all number types	Gives the result of bitwise NOT of A.

Opérateurs intégrés de Hive

La requête suivante ajoute deux nombres, 20 et 30

```
SELECT 20+30 ADD FROM temp;
```

```
+--------+
|  ADD   |
+--------+
|  50    |
+--------+
```

Opérateurs intégrés de Hive

Opérateurs logiques : Les opérateurs sont des expressions logiques. Ils renvoient tous soit VRAI, soit FAUX.

Operators	Operands	Description
A AND B	boolean	TRUE if both A and B are TRUE, otherwise FALSE.
A && B	boolean	Same as A AND B.
A OR B	boolean	TRUE if either A or B or both are TRUE, otherwise FALSE.
A || B	boolean	Same as A OR B.
NOT A	boolean	TRUE if A is FALSE, otherwise FALSE.
!A	boolean	Same as NOT A.

Opérateurs intégrés de Hive

La requête suivante est utilisée pour récupérer les détails de l'employé dont le département est TP et le salaire est supérieur à Rs 40000.

```
SELECT * FROM employee WHERE Salary>40000 && Dept=TP;
```

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
+------+--------------+-------------+-------------------+--------+
```

Opérateurs intégrés de Hive

Opérateurs complexes : Ces opérateurs fournissent une expression pour accéder aux éléments des types complexes.

Operator	Operand	Description
A[n]	A is an Array and n is an int	It returns the nth element in the array A. The first element has index 0.
M[key]	M is a Map<K, V> and key has type K	It returns the value corresponding to the key in the map.
S.x	S is a struct	It returns the x field of S.

Fonctions intégrées de Hive

Hive supporte les fonctions intégrées suivantes :

Return Type	Signature	Description
BIGINT	round(double a)	It returns the rounded BIGINT value of the double.
BIGINT	floor(double a)	It returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	It returns the minimum BIGINT value that is equal or greater than the double.
double	rand(), rand(int seed)	It returns a random number that changes from row to row.
string	concat(string A, string B,...)	It returns the string resulting from concatenating B after A.
string	substr(string A, int start)	It returns the substring of A starting from start position till the end of string A.
string	substr(string A, int start, int length)	It returns the substring of A starting from start position with the given length.
string	upper(string A)	It returns the string resulting from converting all characters of A to upper case.
string	ucase(string A)	Same as above.

Fonctions intégrées de Hive

Hive supporte les fonctions intégrées suivantes :

Return Type	Signature	Description
string	lower(string A)	It returns the string resulting from converting all characters of B to lower case.
string	lcase(string A)	Same as above.
string	trim(string A)	It returns the string resulting from trimming spaces from both ends of A.
string	ltrim(string A)	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
string	rtrim(string A)	rtrim(string A) It returns the string resulting from trimming spaces from the end (right hand side) of A.
string	regexp_replace(string A, string B, string C)	It returns the string resulting from replacing all substrings in B that match the Java regular expression syntax with C.
int	size(Map<K.V>)	It returns the number of elements in the map type.
int	size(Array<T>)	It returns the number of elements in the array type.
value of <type>	cast(<expr> as <type>)	It converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) converts the string '1' to it integral representation. A NULL is returned if the conversion does not succeed.

Fonctions intégrées de Hive

Hive supporte les fonctions intégrées suivantes :

Return Type	Signature	Description
string	from_unixtime(int unixtime)	convert the number of seconds from Unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00"
string	to_date(string timestamp)	It returns the date part of a timestamp string: to_date("1970-01-01 00:00:00") = "1970-01-01"
int	year(string date)	It returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970
int	month(string date)	It returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11
int	day(string date)	It returns the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1
string	get_json_object(string json_string, string path)	It extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It returns NULL if the input json string is invalid.

Fonctions intégrées de Hive

Les requêtes suivantes démontrent certaines fonctions intégrées :

- `SELECT round(2.6) from temp;`

Après l'exécution réussie de la requête, vous obtenez la réponse suivante :

3.0

- `SELECT floor(2.6) from temp`

Après l'exécution réussie de la requête, vous obtenez la réponse suivante :

2.0

- `SELECT ceil(2.6) from temp;`

Après l'exécution réussie de la requête, vous obtenez la réponse suivante :

3.0

Fonctions intégrées de Hive

Hive supporte les fonctions d'agrégation intégrées suivantes. L'utilisation de ces fonctions est la même que celle des fonctions d'agrégation SQL.

Return Type	Signature	Description
BIGINT	count(*), count(expr),	count(*) - Returns the total number of retrieved rows.
DOUBLE	sum(col), sum(DISTINCT col)	It returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg(col), avg(DISTINCT col)	It returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min(col)	It returns the minimum value of the column in the group.
DOUBLE	max(col)	It returns the maximum value of the column in the group.

Index & Vues

Les vues sont générées en fonction des besoins de l'utilisateur. Vous pouvez sauvegarder n'importe quel ensemble de données de résultat comme une vue. L'utilisation de la vue dans Hive est la même que celle de la vue en SQL. Il s'agit d'un concept standard de SGBDR. Nous pouvons exécuter toutes les opérations DML sur une vue.

Vous pouvez créer une vue au moment de l'exécution d'une instruction SELECT.
La syntaxe est la suivante :

```
CREATE VIEW [IF NOT EXISTS] view_name
[(column_name [COMMENT
column_comment], ...) ]
[COMMENT table_comment]
AS SELECT ...
```

Index & Vues

Prenons un exemple de vue. Supposons une table d'employés comme indiqué ci-dessous, avec les champs Id, Nom, Salaire, Désignation, et Département. Générez une requête pour récupérer les détails des employés qui gagnent un salaire de plus de Rs 30000. Nous stockons le résultat dans une vue nommée emp_30000.

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

Index & Vues

- La requête suivante récupère les détails de l'employé :

```
CREATE VIEW emp_30000 AS
SELECT * FROM employee
WHERE salary>30000;
```

- Utilisez la syntaxe suivante pour supprimer une vue :

```
DROP VIEW view_name
```

- La requête suivante supprime une vue nommée emp_30000:

```
DROP VIEW emp_30000;
```

Index & Vues

- Un index n'est rien d'autre qu'un pointeur sur une colonne particulière d'une table.

Créer un index signifie créer un pointeur sur une colonne particulière d'une table.

Index & Vues

```
CREATE INDEX index_name ON TABLE
base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES
(property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
   [ ROW FORMAT ...] STORED AS ...
   | STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
```

Index & Vues

- Prenons un exemple d'index. Utilisez la même table d'employés que nous avons utilisée précédemment avec les champs Id, Name, Salary, Designation, et Dept. Créez un index nommé index_salary sur la colonne de salaire de la table d'employés.
- La requête suivante crée un index :

```
CREATE INDEX index_salary ON TABLE
employee(salary) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIn
dexHandler';
```

Il s'agit d'un pointeur vers la colonne des salaires. Si la colonne est modifiée, les changements sont stockés en utilisant une valeur d'index.

Index & Vues

- La syntaxe suivante est utilisée pour supprimer un index :

```
DROP INDEX <index_name> ON <table_name>
```

- La requête suivante supprime un index nommé index_salary :

```
DROP INDEX index_salary ON employee;
```

HiveQL : Select … Where …

- Le langage de requête Hive (HiveQL) est un langage de requête pour Hive permettant de traiter et d'analyser des données structurées dans un Metastore.
- L'instruction SELECT est utilisée pour récupérer les données d'une table. La clause WHERE fonctionne comme une condition. Elle filtre les données à l'aide de la condition et vous donne un résultat fini. Les opérateurs et fonctions intégrés génèrent une expression qui remplit la condition

```
SELECT [ALL | DISTINCT] select_expr,
select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY
col_list] [SORT BY col_list]]
[LIMIT number];
```

HiveQL : Select … Where …

- Prenons un exemple de clause SELECT…WHERE. Supposons que nous ayons la table des employés comme indiqué ci-dessous, avec des champs nommés Id, Nom, Salaire, Désignation, et Département. Générez une requête pour récupérer les détails des employés qui gagnent un salaire de plus de 30000 Rs.

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

HiveQL : Select … Where …

- La requête suivante récupère les détails de l'employé en utilisant le scénario ci-dessus :

```
SELECT * FROM employee WHERE salary>30000;
```

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
+------+--------------+-------------+-------------------+--------+
```

HiveQL : Select … Where …

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveQLWhere {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      Resultset res = stmt.executeQuery("SELECT * FROM employee WHERE
salary>30000;");
      System.out.println("Result:");
      System.out.println(" ID \t Name \t Salary \t Designation \t Dept ");
      while (res.next()) {
         System.out.println(res.getInt(1) + " " + res.getString(2) + " " +
res.getDouble(3) + " " + res.getString(4) + " " + res.getString(5));
      }
      con.close();
   }
}
```

HiveQL : Select … Where …

Enregistrez le programme dans un fichier nommé HiveQLWhere.java.Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveQLWhere.java
$ java HiveQLWhere
```

Output:

```
ID      Name          Salary      Designation          Dept
1201    Gopal         45000       Technical manager    TP
1202    Manisha       45000       Proofreader          PR
1203    Masthanvali   40000       Technical writer     TP
1204    Krian         40000       Hr Admin             HR
```

HiveQL : Select ….ORDER BY

La clause ORDER BY est utilisée pour récupérer les détails sur la base d'une colonne et trier l'ensemble des résultats par ordre croissant ou décroissant.

```
SELECT [ALL | DISTINCT] select_expr,
select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

HiveQL : Select ….ORDER BY

Prenons un exemple pour la clause SELECT...ORDER BY.
Supposons que le tableau des employés est donné ci-dessous, avec les champs Id, Nom, Salaire, Désignation, et Département.
Générez une requête pour récupérer les détails de l'employé dans l'ordre en utilisant le nom du département.

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

HiveQL : Select ….ORDER BY

La requête suivante récupère les détails de l'employé:

```
SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
```

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
+------+--------------+-------------+-------------------+--------+
```

HiveQL : Select … ORDER BY …

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveQLOrderBy {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
   public static void main(String[] args) throws SQLException {
      Class.forName(driverName);
      Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
      Statement stmt = con.createStatement();
      Resultset res = stmt.executeQuery("SELECT * FROM employee ORDER BY DEPT;");
      System.out.println(" ID \t Name \t Salary \t Designation \t Dept ");
      while (res.next()) {
         System.out.println(res.getInt(1) + " " + res.getString(2) + " " +
res.getDouble(3) + " " + res.getString(4) + " " + res.getString(5));
      }

      con.close();
   }
}
```

HiveQL : Select … ORDER BY …

Enregistrez le programme dans un fichier nommé HiveQLOrderBye.java.Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveQLOrderBy.java
$ java HiveQLOrderBy
```

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
+------+--------------+-------------+-------------------+--------+
```

HiveQL : Select … GROUP BY …

La clause GROUP BY est utilisée pour regrouper tous les enregistrements d'un ensemble de résultats en utilisant une colonne de collecte particulière. Elle est utilisée pour interroger un groupe d'enregistrements.

```
SELECT [ALL | DISTINCT] select_expr,
select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

HiveQL : Select … GROUP BY …

Prenons un exemple de clause SELECT...GROUP BY.
Supposons que la table des employés soit la suivante,
avec les champs Id, Name, Salary, Désignation et Dept.
Générez une requête pour récupérer le nombre d'employés
dans chaque département.

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 45000       | Proofreader       | PR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

HiveQL : Select … GROUP BY …

La requête suivante récupère les détails de l'employé :

```
SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

```
+------+--------------+
| Dept | Count(*)     |
+------+--------------+
|Admin |    1         |
|PR    |    2         |
|TP    |    3         |
+------+--------------+
```

HiveQL : Select … GROUP BY …

Programme JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveQLGroupBy {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

   public static void main(String[] args) throws SQLException {

      // Register driver and create driver instance
      Class.forName(driverName);

      // get connection
      Connection con = DriverManager.
      getConnection("jdbc:hive://localhost:10000/userdb", "", "");

      // create statement
      Statement stmt = con.createStatement();
```

HiveQL : Select … GROUP BY …

Programme JDBC

```
 // execute statement
      Resultset res = stmt.executeQuery("SELECT Dept,count(*) " + "FROM employee
GROUP BY DEPT; ");
      System.out.println(" Dept \t count(*)");

      while (res.next()) {
         System.out.println(res.getString(1) + " " + res.getInt(2));
      }
      con.close();
   }
```

HiveQL : Select … GROUP BY …

Enregistrez le programme dans un fichier nommé HiveQLGroupBy.java.Les commandes suivantes sont utilisées pour compiler et exécuter ce programme

```
$ javac HiveQLGroupBy.java
$ java HiveQLGroupBy
```

Output:

```
Dept      Count(*)
Admin        1
PR           2
TP           3
```

HiveQL : Select Joins

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database.

```
join_table:

    table_reference JOIN table_factor
[join_condition]
    | table_reference {LEFT|RIGHT|FULL}
[OUTER] JOIN table_reference
    join_condition
    | table_reference LEFT SEMI JOIN
table_reference join_condition
    | table_reference CROSS JOIN
table_reference [join_condition]
```

HiveQL : Select Joins

Considérons la table suivante nommée CLIENTS...

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

HiveQL : Select Joins

Considérons une autre table ORDERS comme suit :

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 | 3000   |
| 100 | 2009-10-08 00:00:00 |           3 | 1500   |
| 101 | 2009-11-20 00:00:00 |           2 | 1560   |
| 103 | 2008-05-20 00:00:00 |           4 | 2060   |
+-----+---------------------+-------------+--------+
```

HiveQL : Select Joins

Il existe plusieurs types de « JOIN » :
•JOIN
•LEFT OUTER JOIN
•RIGHT OUTER JOIN
•FULL OUTER JOIN

HiveQL : Select Joins

La clause JOIN est utilisée pour combiner et récupérer les enregistrements de plusieurs tables. JOIN est identique à OUTER JOIN en SQL.
Une condition JOIN doit être posée en utilisant les clés primaires et les clés étrangères des tables.

HiveQL : Select Joins

La requête suivante exécute la clause JOIN sur les tables CUSTOMER et ORDER, et récupère les enregistrements :

```
SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

HiveQL : Select Joins

Le LEFT OUTER JOIN de HiveQL renvoie toutes les lignes de la table de gauche, même s'il n'y a pas de correspondance dans la table de droite. Cela signifie que si la clause ON correspond à 0 (zéro) enregistrement dans la table de droite, le JOIN renvoie toujours une ligne dans le résultat, mais avec NULL dans chaque colonne de la table de droite.

Un LEFT JOIN renvoie toutes les valeurs de la table de gauche, plus les valeurs correspondantes de la table de droite, ou NULL en cas d'absence de prédicat JOIN correspondant.

HiveQL : Select Joins

La requête suivante illustre un LEFT OUTER JOIN entre les tables CUSTOMER et ORDER :

```
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c
LEFT OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

HiveQL : Select Joins

Le RIGHT OUTER JOIN de HiveQL renvoie toutes les lignes de la table de droite, même s'il n'y a pas de correspondance dans la table de gauche. Si la clause ON correspond à 0 (zéro) enregistrement dans la table de gauche, le JOIN renvoie toujours une ligne dans le résultat, mais avec NULL dans chaque colonne de la table de gauche.

Un JOIN DROIT renvoie toutes les valeurs de la table de droite, plus les valeurs correspondantes de la table de gauche, ou NULL en cas d'absence de prédicat de jointure correspondant.

HiveQL : Select Joins

La requête suivante illustre « RIGHT OUTER JOIN » entre les tables CUSTOMER et ORDER.

```
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT
OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID) ;
```

Une fois la requête exécutée avec succès, vous obtenez la réponse suivante :

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

HiveQL : Select Joins

Le FULL OUTER JOIN de HiveQL combine les enregistrements des tables externes de gauche et de droite qui remplissent la condition JOIN. La table jointe contient soit tous les enregistrements des deux tables, soit remplit les valeurs NULL pour les correspondances manquantes des deux côtés.

HiveQL : Select Joins

La requête suivante illustre « FULL OUTER JOIN » entre les tables CUSTOMER et ORDER.

```
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c
FULL OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

HiveQL : Select Joins

Une fois la requête exécutée avec succès, vous obtenez la réponse suivante :

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 1    | Ramesh   | NULL   | NULL                |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5    | Hardik   | NULL   | NULL                |
| 6    | Komal    | NULL   | NULL                |
| 7    | Muffy    | NULL   | NULL                |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

CLI

CLI : command-line interface

Pour obtenir l'aide de Hive
"hive -H" or "hive --help" dans le bin de l'installation de Hive

```
usage: hive
 -d,--define <key=value>          Variable substitution to apply to Hive
                                  commands. e.g. -d A=B or --define A=B
 -e <quoted-query-string>         SQL from command line
 -f <filename>                    SQL from files
 -H,--help                        Print help information
 -h <hostname>                    Connecting to Hive Server on remote host
    --hiveconf <property=value>   Use value for given property
    --hivevar <key=value>         Variable substitution to apply to hive
                                  commands. e.g. --hivevar A=B
 -i <filename>                    Initialization SQL file
 -p <port>                        Connecting to Hive Server on port number
 -S,--silent                      Silent mode in interactive shell
 -v,--verbose                     Verbose mode (echo executed SQL to the
                                  console)
```

CLI

Exemple d'exécution d'une requête à partir de la ligne de commande

```
$HIVE_HOME/bin/hive -e 'select a.col from tab1 a';
```

Exemple de définition de variables de configuration Hive

```
$HIVE_HOME/bin/hive -e 'select a.col from tab1 a' --hiveconf
hive.exec.scratchdir=/home/my/hive_scratch  --hiveconf
mapred.reduce.tasks=32;
```

Exemple d'exécution d'un script non interactif à partir d'un disque local

```
$HIVE_HOME/bin/hive -f /home/my/hive-script.sql;
```

Installation Hive

Telecharger apache-hive-2.1.0-bin.tar.gz

Décompresser dans /usr/local

Installation Hive

Ouvrez ~/.bashrc et définissez la variable d'environnement HIVE_HOME pour qu'elle pointe vers le répertoire d'installation et PATH:

```
export HIVE_HOME=/usr/local/apache-hive-2.1.0-bin
export HIVE_CONF_DIR=/usr/local/apache-hive-2.1.0-bin/conf
export PATH=$HIVE_HOME/bin:$PATH
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/apache-hive-2.1.0-bin/lib/*:.
```

Sourcer le fichier bashrc

```
hduser@Hadoop:~$ source ~/.bashrc
```