**Unit-6**

**HIVE**

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

This is a brief tutorial that provides an introduction on how to use Apache Hive HiveQL with Hadoop Distributed File System. This tutorial can be your first step towards becoming a successful Hadoop Developer with Hive.

Introduction

The term ‘Big Data’ is used for collections of large datasets that include huge volume, high velocity, and a variety of data that is increasing day by day. Using traditional data management systems, it is difficult to process Big Data. Therefore, the Apache Software Foundation introduced a framework called Hadoop to solve Big Data management and processing challenges.

Hadoop

Hadoop is an open-source framework to store and process Big Data in a distributed environment. It contains two modules, one is MapReduce and another is Hadoop Distributed File System (HDFS).

* **MapReduce:** It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware.
* **HDFS:**Hadoop Distributed File System is a part of Hadoop framework, used to store and process the datasets. It provides a fault-tolerant file system to run on commodity hardware.

The Hadoop ecosystem contains different sub-projects (tools) such as Sqoop, Pig, and Hive that are used to help Hadoop modules.

* **Sqoop:** It is used to import and export data to and from between HDFS and RDBMS.
* **Pig:** It is a procedural language platform used to develop a script for MapReduce operations.
* **Hive:** It is a platform used to develop SQL type scripts to do MapReduce operations.

**Note:** There are various ways to execute MapReduce operations:

* The traditional approach using Java MapReduce program for structured, semi-structured, and unstructured data.
* The scripting approach for MapReduce to process structured and semi structured data using Pig.
* The Hive Query Language (HiveQL or HQL) for MapReduce to process structured data using Hive.

What is Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

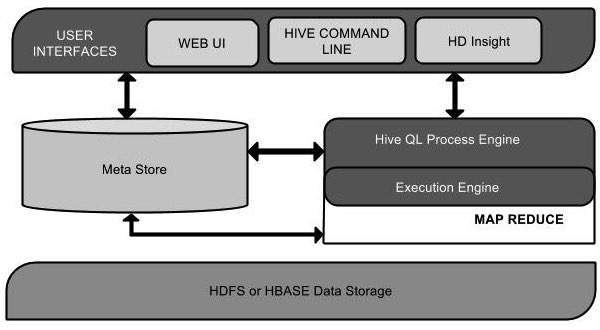
* A relational database
* A design for OnLine Transaction Processing (OLTP)
* A language for real-time queries and row-level updates

Features of Hive

* It stores schema in a database and processed data into HDFS.
* It is designed for OLAP.
* It provides SQL type language for querying called HiveQL or HQL.
* It is familiar, fast, scalable, and extensible.

Architecture of Hive

The following component diagram depicts the architecture of Hive:



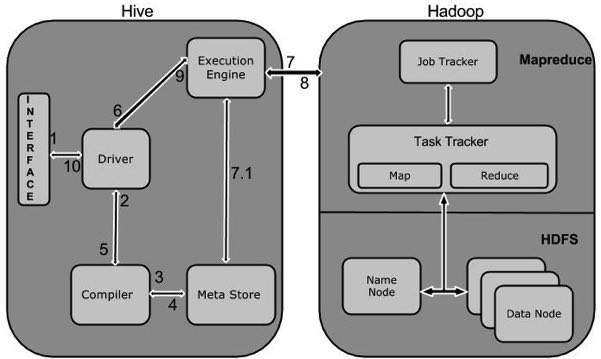
This component diagram contains different units. The following table describes each unit:

This component diagram contains different units. The following table describes each unit:

|  |  |
| --- | --- |
| **Unit Name** | **Operation** |
| User Interface | Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server). |
| Meta Store | Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping. |
| HiveQL Process Engine | HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it. |
| Execution Engine | The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce. |
| HDFS or HBASE | Hadoop distributed file system or HBASE are the data storage techniques to store data into file system. |

## Working of Hive

The following diagram depicts the workflow between Hive and Hadoop



# Hive - Data Types

This chapter takes you through the different data types in Hive, which are involved in the table creation. All the data types in Hive are classified into four types, given as follows:

* Column Types
* Literals
* Null Values
* Complex Types

Column Types

Column type are used as column data types of Hive. They are as follows:

Integral Types

Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

|  |  |  |
| --- | --- | --- |
| **Type** | **Postfix** | **Example** |
| TINYINT | Y | 10Y |
| SMALLINT | S | 10S |
| INT | - | 10 |
| BIGINT | L | 10L |

String Types

String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

The following table depicts various CHAR data types:

|  |  |
| --- | --- |
| **Data Type** | **Length** |
| VARCHAR | 1 to 65355 |
| CHAR | 255 |

Timestamp

It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” and format “yyyy-mm-dd hh:mm:ss.ffffffffff”.

Dates

DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimals

The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

DECIMAL(precision, scale)

decimal(10,0)

Union Types

Union is a collection of heterogeneous data types. You can create an instance using **create union**. The syntax and example is as follows:

UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

{0:1}

{1:2.0}

{2:["three","four"]}

{3:{"a":5,"b":"five"}}

{2:["six","seven"]}

{3:{"a":8,"b":"eight"}}

{0:9}

{1:10.0}

Literals

The following literals are used in Hive:

Floating Point Types

Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type

Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10-308 to 10308.

Null Value

Missing values are represented by the special value NULL.

Complex Types

The Hive complex data types are as follows:

Arrays

Arrays in Hive are used the same way they are used in Java.

Syntax: ARRAY<data\_type>

Maps

Maps in Hive are similar to Java Maps.

Syntax: MAP<primitive\_type, data\_type>

Structs

Structs in Hive is similar to using complex data with comment.

Syntax: STRUCT<col\_name : data\_type [COMMENT col\_comment], ...>

# Hive - Create Database

Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it. This chapter explains how to create Hive database. Hive contains a default database named **default**.

## Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

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

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

hive> CREATE DATABASE [IF NOT EXISTS] userdb;

**or**

hive> CREATE SCHEMA userdb;

The following query is used to verify a databases list:

hive> SHOW DATABASES;

default

userdb

### JDBC Program

The JDBC program to create a database is given below.

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();

}

}

Save the program in a file named HiveCreateDb.java. The following commands are used to compile and execute this program.

$ javac HiveCreateDb.java

$ java HiveCreateDb

### Output:

Database userdb created successfully.

# Hive - Drop Database

## Drop Database Statement

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database\_name

[RESTRICT|CASCADE];

The following queries are used to drop a database. Let us assume that the database name is **userdb**.

hive> DROP DATABASE IF EXISTS userdb;

The following query drops the database using **CASCADE**. It means dropping respective tables before dropping the database.

hive> DROP DATABASE IF EXISTS userdb CASCADE;

The following query drops the database using **SCHEMA**.

hive> DROP SCHEMA userdb;

This clause was added in Hive 0.6.

### JDBC Program

The JDBC program to drop a database is given below.

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();

}

}

Save the program in a file named HiveDropDb.java. Given below are the commands to compile and execute this program.

$ javac HiveDropDb.java

$ java HiveDropDb

### Output:

Drop userdb database successful.

# Hive - Create Table

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

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]

Example

Let us assume you need to create a table named **employee** using **CREATE TABLE** statement. The following table lists the fields and their data types in employee table:

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

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

COMMENT ‘Employee details’

FIELDS TERMINATED BY ‘\t’

LINES TERMINATED BY ‘\n’

STORED IN TEXT FILE

The following query creates a table named **employee** using the above data.

hive> 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;

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

On successful creation of table, you get to see the following response:

OK

Time taken: 5.905 seconds

hive>

JDBC Program

The JDBC program to create a table is given example.

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 {

// 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();

// execute statement

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();

}

}

Save the program in a file named HiveCreateDb.java. The following commands are used to compile and execute this program.

$ javac HiveCreateDb.java

$ java HiveCreateDb

Output

Table employee created.

Load Data Statement

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

Syntax

The syntax for load data is as follows:

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename

[PARTITION (partcol1=val1, partcol2=val2 ...)]

* LOCAL is identifier to specify the local path. It is optional.
* OVERWRITE is optional to overwrite the data in the table.
* PARTITION is optional.

Example

We will insert the following data into the table. It is a text file named**sample.txt** in **/home/user** directory.

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

The following query loads the given text into the table.

hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'

OVERWRITE INTO TABLE employee;

On successful download, you get to see the following response:

OK

Time taken: 15.905 seconds

hive>

JDBC Program

Given below is the JDBC program to load given data into the table.

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 {

// 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();

// execute statement

stmt.executeQuery("LOAD DATA LOCAL INPATH '/home/user/sample.txt'" + "OVERWRITE INTO TABLE employee;");

System.out.println("Load Data into employee successful");

con.close();

}

}

Save the program in a file named HiveLoadData.java. Use the following commands to compile and execute this program.

$ javac HiveLoadData.java

$ java HiveLoadData

Output:

Load Data into employee successful

# Hive - Alter Table

## Alter Table Statement

It is used to alter a table in Hive.

### Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

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 ...])

## Rename To… Statement

The following query renames the table from **employee** to **emp**.

hive> ALTER TABLE employee RENAME TO emp;

### JDBC Program

The JDBC program to rename a table is as follows.

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 {

// 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();

// execute statement

stmt.executeQuery("ALTER TABLE employee RENAME TO emp;");

System.out.println("Table Renamed Successfully");

con.close();

}

}

Save the program in a file named HiveAlterRenameTo.java. Use the following commands to compile and execute this program.

$ javac HiveAlterRenameTo.java

$ java HiveAlterRenameTo

### Output:

Table renamed successfully.

## Change Statement

The following table contains the fields of **employee** table and it shows the fields to be changed (in bold).

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

The following queries rename the column name and column data type using the above data:

hive> ALTER TABLE employee CHANGE name ename String;

hive> ALTER TABLE employee CHANGE salary salary Double;

### JDBC Program

Given below is the JDBC program to change a column.

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);

// get connection

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

// create statement

Statement stmt = con.createStatement();

// execute statement

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();

}

}

Save the program in a file named HiveAlterChangeColumn.java. Use the following commands to compile and execute this program.

$ javac HiveAlterChangeColumn.java

$ java HiveAlterChangeColumn

### Output:

Change column successful.

## Add Columns Statement

The following query adds a column named dept to the employee table.

hive> ALTER TABLE employee ADD COLUMNS (

dept STRING COMMENT 'Department name');

## JDBC Program

The JDBC program to add a column to a table is given below.

import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;

public class HiveAlterAddColumn {

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();

// execute statement

stmt.executeQuery("ALTER TABLE employee ADD COLUMNS " + " (dept STRING COMMENT 'Department name');");

System.out.prinln("Add column successful.");

con.close();

}

}

Save the program in a file named HiveAlterAddColumn.java. Use the following commands to compile and execute this program.

$ javac HiveAlterAddColumn.java

$ java HiveAlterAddColumn

### Output:

Add column successful.

## Replace Statement

The following query deletes all the columns from the **employee** table and replaces it with **emp** and **name** columns:

hive> ALTER TABLE employee REPLACE COLUMNS (

eid INT empid Int,

ename STRING name String);

## JDBC Program

Given below is the JDBC program to replace **eid** column with **empid** and**ename**column with **name**.

import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;

public class HiveAlterReplaceColumn {

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();

// execute statement

stmt.executeQuery("ALTER TABLE employee REPLACE COLUMNS "

+" (eid INT empid Int,"

+" ename STRING name String);");

System.out.println(" Replace column successful");

con.close();

}

}

Save the program in a file named HiveAlterReplaceColumn.java. Use the following commands to compile and execute this program.

$ javac HiveAlterReplaceColumn.java

$ java HiveAlterReplaceColumn

### Output:

Replace column successful.

# Hive - Drop Table

## Drop Table Statement

The syntax is as follows:

DROP TABLE [IF EXISTS] table\_name;

The following query drops a table named **employee**:

hive> DROP TABLE IF EXISTS employee;

On successful execution of the query, you get to see the following response:

OK

Time taken: 5.3 seconds

hive>

### JDBC Program

The following JDBC program drops the employee table.

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 {

// 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();

// execute statement

stmt.executeQuery("DROP TABLE IF EXISTS employee;");

System.out.println("Drop table successful.");

con.close();

}

}

Save the program in a file named HiveDropTable.java. Use the following commands to compile and execute this program.

$ javac HiveDropTable.java

$ java HiveDropTable

### Output:

Drop table successful

The following query is used to verify the list of tables:

hive> SHOW TABLES;

emp

ok

Time taken: 2.1 seconds

hive>