Clareus Scientific Science and Engineering
Volume 2 Issue 7 September 2025
ISSN: 3065-1182

HIVE on Hadoop on Ubuntu: A Step-by-Step Guide

Citation: Rupali M Bora. "HIVE on Hadoop on Ubuntu: A Step-by-Step Guide". Clareus Scientific Science and Engineering 2.7 (2025): 59-61.

Article Type: Short Communication

Received: August 23, 2025

Published: September 06, 2025

Rupali M Bora*

Assistant Professor, Department of IT, K. K. Wagh Institute of Engineering Education & Research, Nashik, India

***Corresponding Author:** Rupali M Bora, Assistant Professor, Department of IT, K. K. Wagh Institute of Engineering Education & Research, Nashik, India.

This article provides a comprehensive walkthrough of a series of operations performed in Apache Hive on a Hadoop cluster running on the Ubuntu operating system. It demonstrates fundamental HiveQL commands, including table creation, data loading, data manipulation, schema modification, and indexing, while highlighting the underlying MapReduce job execution. This guide is intended for data engineers and analysts seeking a practical understanding of Hive's functionalities for data warehousing on a Hadoop platform.

Environment Setup and Initial Status

The session begins by launching the Hive shell from the terminal. The output confirms that Hive is successfully initialized. Key messages include:

- ***SLF4J Bindings***: A warning indicating multiple logging bindings were found. This is a common occurrence and doesn't impede functionality, but it's good practice to resolve it for a cleaner setup. Hive defaults to log4j-slf4j-impl-2.4.1.jar in this case.
- ***Hive-on-MR Deprecation***: A prominent warning notes that MapReduce is deprecated as the default execution engine in Hive 2. The output suggests using more modern, faster engines like Spark or Tez for production environments.

The initial checks show databases; and show tables; confirm the default database exists and that no tables are present in it, providing a clean slate for the following operations.

Data Definition Language (DDL) Operations

Creating a Database and Table

First, a new database named db1 is created using the CREATE DATABASE db1; command. While the subsequent operations are performed in the default database, this command showcases a basic DDL capability.

The cust table is then created with two columns: cname (string) and csal (int). It is defined as a delimited text file, a simple and common format in Hive, where fields are separated by a comma. The show tables; command confirms the table's successful creation.

Altering a Table

The schema of the cust table is modified to add a new column, dept of type string. The ALTER TABLE cust ADD COLUMNS (dept string); command performs this change without affecting the existing data. The select * from cust; query after this operation reveals that the new column is populated with NULL values for all existing rows, as expected. The desc cust; command provides a detailed description of the updated table schema.

Dropping a Table

A new table, table_name, is created as an example to demonstrate the DROP TABLE command. This table is a bucketed table, clustered by id into 2 buckets and stored in the ORC (Optimized Row Columnar) format. ORC is a highly efficient, columnar storage format often used in Hive for better compression and faster query performance. The drop table table_name; command successfully removes the table from the Hive Metastore and deletes the associated data from HDFS.

Data Manipulation Language (DML) Operations

Loading and Querying Data

Data is loaded into the cust table from a local text file named cust.txt using LOAD DATA LOCAL INPATH '/home/student/Desktop/cust.txt' INTO TABLE cust;. Hive efficiently loads the data into its managed location on the Hadoop Distributed File System (HDFS). The select * from cust; query confirms that all six rows from the file are correctly loaded. A subsequent select query with a WHERE clause demonstrates basic filtering, returning only the row where csal equals 20000.

Inserting a New Record

The command INSERT INTO TABLE cust VALUES ('shilpa', '65000'); adds a new record to the table. This is a crucial operation that triggers a MapReduce job behind the scenes. The output logs confirm the job execution, including:

- ***Query ID***: A unique identifier for the query.
- ***Total jobs***: The number of MapReduce jobs launched (in this case, 3 jobs are launched to manage the insert operation).
- ***MapReduce job details***: Information on job progress (mapper and reducer percentages), cumulative CPU time, and HDFS read/write statistics. The final select * from cust; query shows the new row for 'shilpa' has been successfully appended.

Advanced Hive Features

Joining Tables

A join operation is performed on two tables, flight1 and schedule, using the fno column as the common key. The query select f.fno, fcity, dcity, s.dat from flight1 f join schedule s on (f.fno=s.fno); retrieves data by combining rows from both tables. The extensive output logs highlight the execution plan, including:

- ***Map Join Optimization***: Hive's execution engine recognizes this as a small-table join and optimizes it using a Map Join, where one table is loaded into memory to significantly speed up the join process.
- ***Local Task Execution***: The logs show a "local task" is launched to prepare the side-table for the map join.
- ***Hadoop Job Execution***: The process culminates in a single MapReduce job that performs the join operation on the distributed data. The final output shows the joined results.

Indexing

Indexing is a performance optimization technique used to speed up data retrieval. A compact index named idx is created on the dat column of the schedule table using CREATE INDEX idx ON TABLE schedule(dat) AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;.

The WITH DEFERRED REBUILD clause indicates that the index will be built later. This is important because, unlike traditional relational databases, Hive's indices are metadata-based and require a separate process to populate the index table. The query select count(dat), dat from schedule group by dat; is executed to demonstrate a simple query after index creation. The output shows the results of the GROUP BY operation.

However, it's worth noting that Hive indexes are largely deprecated in modern versions in favor of more effective techniques like partitioning, bucketing, and using optimized file formats (ORC, Parquet), which provide similar or better performance gains with less maintenance overhead. The subsequent select avg(csal) from cust; queries show that creating an index on cust does not significantly improve the execution time for an aggregation query, reinforcing the limited utility of indexing in this context.

External Tables

The final section demonstrates the creation of an external table that maps to data stored in Apache HBase. The command CREATE EXTERNAL TABLE hbase_table_3(...) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' ... TBLPROPERTIES ("hbase.table.name" = "cust"); links a Hive table schema to an existing HBase table named cust. The SerDe properties (hbase.columns.mapping) are critical here, as they define how Hive's columns (key, value, etc.) map to HBase's row key and column families. The subsequent desc 'employee' and scan 'employee' commands from the HBase shell show the underlying data structure in HBase, which includes column families like persnal and saldet and demonstrates how Hive can be used as a query layer on top of HBase data.