Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department


جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

# Lecture (7 & 8): Hive in the Hadoop Ecosystem

**Subject: Big Data Analysis in Healthcare**
**Level: Fourth**
**Lecturer: Asst. Lecturer Qusai AL-Durrah**
**Duration: Two hours**

## 1. Introduction

Apache Hive is a core component of the Hadoop Ecosystem that provides a **high-level data warehousing solution** on top of the Hadoop Distributed File System (HDFS). Designed initially at Facebook to address the difficulty analysts faced when writing low-level MapReduce programs, Hive introduces a SQL-like language called **HiveQL**, enabling analysts, researchers, and students to process massive datasets using declarative queries rather than procedural code.

Hive converts HiveQL statements into **MapReduce**, **Tez**, or **Spark** jobs, depending on the underlying execution engine. As emphasized by *Tom White (Hadoop: The Definitive Guide, 3rd Ed.)*, Hive's strength lies in its ability to operate on extremely large datasets stored in HDFS while presenting a familiar relational-style query interface. Unlike traditional databases, Hive follows a **Schema-on-Read** principle, allowing flexible ingestion of structured, semi-structured, and unstructured healthcare data.

In healthcare environments—where data is rapidly accumulating from EHR systems, IoT medical devices, imaging modalities, laboratory information systems, and national health registries—Hive plays a central role in enabling large-scale analytical processing for evidence-based healthcare management.

## 2. Learning Objectives

By the end of this lecture, students will be able to:

1. Explain the purpose of Hive within the Hadoop Ecosystem.
2. Describe the internal architecture of Hive (Driver, Compiler, Execution Engine, Metastore).
3. Clarify the differences between Hive and traditional relational databases.
4. Understand Hive's data model (tables, partitions, buckets, file formats).
5. Write and interpret basic HiveQL statements for analytical workloads.

6. Describe how Hive supports large-scale healthcare analytics such as patient statistics, epidemiological trends, and imaging metadata processing.

## 3. Background and Motivation for Hive

### 3.1 The Need for High-Level Data Analysis on Hadoop

Hadoop introduced a powerful distributed file system (HDFS) and a scalable computation model (MapReduce). However, MapReduce is difficult to write and maintain—especially for analysts and researchers who require declarative tools.

Hive bridges this gap by allowing analysts to run SQL-style queries over datasets stored in HDFS. It acts as the **data warehouse layer** of Hadoop, enabling:

- Structured querying
- Aggregation and summarization
- Batch analytical processing
- Metadata management

### 3.2 Design Philosophy

Hive was built with these guiding principles:

- **SQL-Like Accessibility:** HiveQL resembles SQL, reducing the learning curve.
- **Batch Processing:** Optimized for large analytical queries, not real-time results.
- **Extensibility:** Users can define custom serialization/deserialization (SerDe), functions, and storage formats.
- **Separation of Storage and Metadata:** Data is in HDFS; metadata is in a relational Metastore.

### 3.3 Hive in Healthcare Analytics

Hive enables:

- Multi-year patient record analysis
- Hospital performance dashboards
- Epidemiological research at national scale
- Integration with machine learning (through Spark ML pipelines)
- Analysis of imaging metadata and radiology workflow logs
- Predictive modeling with aggregated features

## 4. Hive Architecture

Hive consists of several coordinated components that work together to execute queries over massive datasets.

### 4.1 User Interfaces

Hive supports various interfaces:

- **Hive CLI** – command line interface
- **Beeline** – (Java Database Connectivity) JDBC-based command interface
- **Hue** – web-based GUI
- **JDBC/ODBC(Open Database Connectivity) clients** – integration with BI tools

These interfaces accept HiveQL queries and send them to the Hive driver.

### 4.2 Driver

The driver manages the entire lifecycle of a query:

- Receives the query

- Creates a session handle
- Manages the compilation and optimization process
- Monitors job execution
- Returns results to the user

It acts similarly to the front controller in relational database systems.

### 4.3 Compiler

The Hive compiler performs several steps:

1. **Parsing:** Converts the HiveQL text into an abstract syntax tree.
2. **Semantic Analysis:** Validates table and column references via the Metastore.
3. **Logical Plan Generation:** Converts the Abstract Syntax Tree (AST) into a logical plan of relational operations.
4. **Physical Plan Generation:** Converts the logical plan into an executable DAG of tasks (MapReduce/Tez/Spark).

### 4.4 Metastore

The Metastore is the metadata repository of Hive. It stores:

- Database names
- Table definitions
- Column types
- Partitions
- Bucketing information
- File locations in HDFS

The Metastore is commonly backed by MySQL or PostgreSQL.

### 4.5 Execution Engine

Once the plan is generated, Hive uses an execution engine to run jobs on YARN. Supported engines include:

- **MapReduce** – default in older Hadoop versions
- **Tez** – Directed Acyclic Graph (DAG) based engine for faster processing
- **Spark** – optimized for iterative and interactive queries

## 5. Hive Data Model

Hive provides a flexible data model suited for large-scale analytical workloads.

### 5.1 Databases

A Hive database is a logical namespace containing tables. Healthcare institutions may define databases such as:

- ehr_data,
- imaging_registry,
- hospital_operations.

### 5.2 Tables

Hive tables correspond to directories in HDFS. Two types:

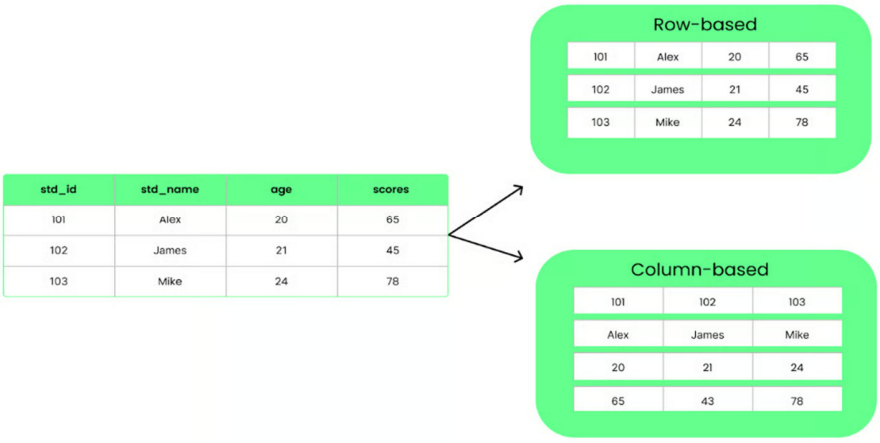
- **Managed Tables:** Hive manages both data and metadata.
- **External Tables:** Only metadata is managed; data stays in external locations (useful for imaging metadata and third-party datasets).

### 5.3 File Formats

Hive supports several storage formats:

| Format | Description | Healthcare Usage |
|---|---|---|
| **TEXTFILE** | Default, human-readable | Raw logs, unprocessed data |
| **SEQUENCEFILE** | Binary, splittable | High-throughput ingestion |
| **ORC** | Optimized row-column format | EHR datasets with frequent queries |
| **Parquet** | Columnar format | Imaging metadata, analytics workloads |

Columnar formats (ORC, Parquet) are preferred for analytical workloads because they reduce I/O and improve query execution.




## 6. HiveQL Language (Ambari-Based Practical Workflow)

This section introduces HiveQL from a practical perspective suitable for working within the Ambari environment. Ambari provides a centralized interface for interacting with Hive and allows users to:

- Create and manage Hive tables
- Insert and query data

- Inspect metadata
- Execute HiveQL statements
- View results and monitor query execution

To maintain consistency throughout the lecture, we will use a single example table—**patient_visits**—and demonstrate HiveQL concepts through step-by-step SQL commands that reflect typical analytical operations on healthcare datasets.

All SQL statements in this section can be executed directly within:

**Ambari → Hive → Query Editor**

### 6.1 Creating a Basic Table

Hive tables can be created using the CREATE TABLE statement. Below is a simple schema for storing patient visit records:

```
CREATE TABLE patient_visits (
    patient_id     INT,
    visit_date     STRING,
    hospital_id    STRING,
    diagnosis_code STRING,
    cost           DOUBLE
);
```

This structure is sufficient for analytical queries over medical visits, hospital operations, and diagnostic trends.

### 6.2 Inserting Sample Records

Hive allows inserting data directly using SQL syntax:

```
INSERT INTO patient_visits VALUES
(101, '2023-01-11', 'HSP01', 'J10.1', 45.5),
(102, '2023-02-05', 'HSP02', 'E11.9', 70.0),
(101, '2023-02-18', 'HSP01', 'J10.1', 55.0),
(103, '2023-03-01', 'HSP03', 'I10', 33.0),
(104, '2023-03-04', 'HSP02', 'E11.9', 110.0);
```

Additional rows can be inserted at any time to expand the analytical dataset.

**6.3 Querying the Table**

HiveQL supports the common SELECT operations used for analysis.

**Retrieve all records:**

```
SELECT * FROM patient_visits;
```

**Filtering results:**

```
SELECT *
FROM patient_visits
WHERE diagnosis_code = 'E11.9';
```

**Selecting specific columns:**

```
SELECT patient_id, diagnosis_code, cost
FROM patient_visits;
```

**6.4 Aggregation and Analytical Queries**

Hive is optimized for analytical workloads such as aggregations over large datasets.

**Count visits per diagnosis:**

SELECT diagnosis_code, COUNT(*) AS total_visits

FROM patient_visits

GROUP BY diagnosis_code;

**Average cost per hospital:**

SELECT hospital_id, AVG(cost) AS avg_cost

FROM patient_visits

GROUP BY hospital_id;

These queries demonstrate Hive's ability to summarize patient activity, treatment costs, and diagnostic distributions.

### 6.5 Extending the Dataset

More data can be inserted to simulate continuous growth:

INSERT INTO patient_visits VALUES

(105, '2023-04-10', 'HSP01', 'I10',  40.0),

(106, '2023-04-12', 'HSP03', 'J10.1', 60.0);

Re-running the analytical queries reflects the updated dataset, illustrating Hive's incremental data-processing model.

### 6.6 Joining with Reference Data

Hive supports joining analytical fact tables with reference (dimension) tables.

**Create a small diagnosis reference table:**

CREATE TABLE diagnosis_reference (

  code  STRING,

  description STRING

);

**Insert reference descriptions:**

INSERT INTO diagnosis_reference VALUES

('J10.1', 'Influenza with respiratory symptoms'),

('E11.9', 'Type 2 diabetes mellitus'),

('I10', 'Essential hypertension');

**Join operation:**

SELECT pv.patient_id, pv.visit_date, dr.description, pv.cost

FROM patient_visits pv

JOIN diagnosis_reference dr

ON pv.diagnosis_code = dr.code;

This type of join is common in healthcare analytics when linking visit-level facts with medical coding standards.

**6.7 Modifying Data (Overwrite Semantics)**

Hive does not support row-level updates in typical TEXTFILE tables. Instead, modifications are performed using *overwrite* operations.

**Example: Adjusting cost values:**

INSERT OVERWRITE TABLE patient_visits

SELECT patient_id, visit_date, hospital_id, diagnosis_code,

CASE WHEN patient_id = 101 THEN 50.0 ELSE cost END

FROM patient_visits;

**Deleting specific rows:**

```
INSERT OVERWRITE TABLE patient_visits
SELECT *
FROM patient_visits
WHERE patient_id <> 104;
```

These examples demonstrate Hive's batch-processing nature.

**7. Hive Workflow in the Hadoop Ecosystem**

Typical workflow:

1. **Data Ingestion:**
   Using Flume, Kafka, or Sqoop to ingest data into HDFS.
2. **Storage:**
   Data stored in HDFS in raw or optimized formats.
3. **Metadata Registration:**
   Create Hive tables using CREATE TABLE.
4. **Execution:**
   Hive translates queries into distributed jobs executed via YARN.
5. **Analysis & Reporting:**
   Output consumed by BI tools or exported to dashboards.

**8. Hive in Healthcare: Practical Applications**

**8.1 Electronic Health Record (EHR) Analytics**

- Outpatient/inpatient statistics
- Readmission analysis
- Treatment outcome grouping
- Longitudinal patient tracking

### 8.2 Medical Imaging Metadata Warehouse

Hive enables the storage and analysis of metadata such as:

- Study ID
- Modality (CT, MRI, X-ray)
- Body region
- Technician
- Acquisition time

Useful for:

- Radiology workflow optimization
- Peak-time analysis
- Procedure demand forecasting

### 8.3 Public Health and Epidemiology

Hive supports:

- Case count aggregation
- Spatiotemporal disease spread analysis
- Risk factor correlations
- Vaccination statistics

### 8.4 Hospital Operational Intelligence

Examples:

- ED wait time analytics

- Bed occupancy forecasting
- Cost-of-care analysis
- Staff workload metrics

## 9. Advantages and Limitations of Hive

### 9.1 Advantages

- Familiar SQL interface for analysts
- Highly scalable and fault-tolerant
- Integrates well with HDFS and YARN
- Supports multiple execution engines
- Efficient analytical processing using ORC/Parquet

### 9.2 Limitations

- Not suitable for OLTP workloads
- Higher query latency
- Requires careful design of partitions and schemas
- Dependent on underlying cluster performance

## 10. Summary

Hive is a foundational element of the Hadoop Ecosystem, providing a scalable, flexible, SQL-like environment for processing massive datasets. Through its integration with HDFS, YARN, and advanced file formats, Hive enables healthcare institutions to transform raw data into meaningful insights, supporting evidence-based clinical practice, operational efficiency, and public health research.