(Big) Data Engineering In Depth

Moustafa Mahmoud
Data Solution Architect

Chapter: Hive

Chapter Objectives

- Query data lakes/HDFS using Hive.

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive
- Comparing Hive to Traditional databases.

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive
- Comparing Hive to Traditional databases.
- Hive Components and Architecture.

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive
- Comparing Hive to Traditional databases.
- Hive Components and Architecture.
- Relational Data Analysis with Hive.

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive
- Comparing Hive to Traditional databases.
- Hive Components and Architecture.
- Relational Data Analysis with Hive.
- Hive Data Management.

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive
- Comparing Hive to Traditional databases.
- Hive Components and Architecture.
- Relational Data Analysis with Hive.
- Hive Data Management.
- Hive Optimization.

Chapter Objectives

- Query data lakes/HDFS using Hive.
- Introduction to Hive
- Comparing Hive to Traditional databases.
- Hive Components and Architecture.
- Relational Data Analysis with Hive.
- Hive Data Management.
- Hive Optimization.
- Hive Demo.

Section: Query data lakes/HDFS using Hive

Query data lakes/HDFS using Hive

- Data lakes and HDFS store vast amounts of petabyte-scale data.

Query data lakes/HDFS using Hive

- Data lakes and HDFS store vast amounts of petabyte-scale data.
- Data analysts require efficient ways to query this data without delving into intricate Map-Reduce programming.

Query data lakes/HDFS using Hive

- Data lakes and HDFS store vast amounts of petabyte-scale data.
- Data analysts require efficient ways to query this data without delving into intricate Map-Reduce programming.
- SQL is a commonly used language for data manipulation, embraced by data analysts, developers, and business users worldwide.

Query data lakes/HDFS using Hive

- Data lakes and HDFS store vast amounts of petabyte-scale data.
- Data analysts require efficient ways to query this data without delving into intricate Map-Reduce programming.
- SQL is a commonly used language for data manipulation, embraced by data analysts, developers, and business users worldwide.
- Hive, one of the early tools in this field, simplifies data analysis on big data by offering SQL-like querying capabilities for data lakes and HDFS.

Query data lakes/HDFS using Hive

- Data lakes and HDFS store vast amounts of petabyte-scale data.
- Data analysts require efficient ways to query this data without delving into intricate Map-Reduce programming.
- SQL is a commonly used language for data manipulation, embraced by data analysts, developers, and business users worldwide.
- Hive, one of the early tools in this field, simplifies data analysis on big data by offering SQL-like querying capabilities for data lakes and HDFS.
- There are other tools like Trino/Presto, Snowflake, and Databricks that can be faster for data lake queries, which we will discuss later.

Query data lakes/HDFS using Hive

- Data lakes and HDFS store vast amounts of petabyte-scale data.
- Data analysts require efficient ways to query this data without delving into intricate Map-Reduce programming.
- SQL is a commonly used language for data manipulation, embraced by data analysts, developers, and business users worldwide.
- Hive, one of the early tools in this field, simplifies data analysis on big data by offering SQL-like querying capabilities for data lakes and HDFS.
- There are other tools like Trino/Presto, Snowflake, and Databricks that can be faster for data lake queries, which we will discuss later.

Section: Introduction to hive

Introduction to hive

- Apache Hive is a powerful data warehousing and SQL-like query language tool within the Hadoop ecosystem.

Introduction to hive

- Apache Hive is a powerful data warehousing and SQL-like query language tool within the Hadoop ecosystem.
- It was developed by Facebook and later contributed to the Apache Software Foundation, making it an open-source project.

Introduction to hive

- Apache Hive is a powerful data warehousing and SQL-like query language tool within the Hadoop ecosystem.
- It was developed by Facebook and later contributed to the Apache Software Foundation, making it an open-source project.
- Apache Hive had its initial release on October 1, 2010.

Introduction to hive

- Apache Hive is a powerful data warehousing and SQL-like query language tool within the Hadoop ecosystem.
- It was developed by Facebook and later contributed to the Apache Software Foundation, making it an open-source project.
- Apache Hive had its initial release on October 1, 2010.

Introduction to hive

- Hive provides a user-friendly interface to work with and analyze large datasets stored in Hadoop Distributed File System (HDFS) or other compatible storage systems.

Introduction to hive

- Hive provides a user-friendly interface to work with and analyze large datasets stored in Hadoop Distributed File System (HDFS) or other compatible storage systems.
- With its SQL-like language called HiveQL, users can write queries to extract, transform, and analyze data, making it accessible to a wide range of data professionals

Introduction to hive

- Hive provides a user-friendly interface to work with and analyze large datasets stored in Hadoop Distributed File System (HDFS) or other compatible storage systems.
- With its SQL-like language called HiveQL, users can write queries to extract, transform, and analyze data, making it accessible to a wide range of data professionals
- Apache Hive bridges the gap between the world of big data and traditional relational databases, making it a valuable tool for data engineers, analysts, and data scientists.

Introduction to hive

- Hive provides a user-friendly interface to work with and analyze large datasets stored in Hadoop Distributed File System (HDFS) or other compatible storage systems.
- With its SQL-like language called HiveQL, users can write queries to extract, transform, and analyze data, making it accessible to a wide range of data professionals
- Apache Hive bridges the gap between the world of big data and traditional relational databases, making it a valuable tool for data engineers, analysts, and data scientists.

Hive Query Example

```
1 SELECT *
2 FROM Customers
3 WHERE Country = 'USA';
```

Code Snippet: Hive Query Example

Sub-Section: Overview of Apache Hive

Overview of Apache Hive

- Apache Hive is a high-level data warehousing and SQL-like query language tool built on top of MapReduce.

Overview of Apache Hive

- Apache Hive is a high-level data warehousing and SQL-like query language tool built on top of MapReduce.
- It acts as an abstraction layer, allowing users to work with large datasets stored in Hadoop Distributed File System (HDFS) without needing to write complex MapReduce jobs themselves.

Overview of Apache Hive

- Apache Hive is a high-level data warehousing and SQL-like query language tool built on top of MapReduce.
- It acts as an abstraction layer, allowing users to work with large datasets stored in Hadoop Distributed File System (HDFS) without needing to write complex MapReduce jobs themselves.
- Under the hood, Hive generates MapReduce jobs that are executed on the Hadoop cluster. This means it leverages the distributed processing capabilities of Hadoop for scalability and parallel processing.

Overview of Apache Hive

- Apache Hive is a high-level data warehousing and SQL-like query language tool built on top of MapReduce.
- It acts as an abstraction layer, allowing users to work with large datasets stored in Hadoop Distributed File System (HDFS) without needing to write complex MapReduce jobs themselves.
- Under the hood, Hive generates MapReduce jobs that are executed on the Hadoop cluster. This means it leverages the distributed processing capabilities of Hadoop for scalability and parallel processing.
- Hive has evolved beyond MapReduce to support other execution engines, such as Tez and Spark, which can enhance performance and usability. This will be discussed later.

Overview of Apache Hive

- Apache Hive is a high-level data warehousing and SQL-like query language tool built on top of MapReduce.
- It acts as an abstraction layer, allowing users to work with large datasets stored in Hadoop Distributed File System (HDFS) without needing to write complex MapReduce jobs themselves.
- Under the hood, Hive generates MapReduce jobs that are executed on the Hadoop cluster. This means it leverages the distributed processing capabilities of Hadoop for scalability and parallel processing.
- Hive has evolved beyond MapReduce to support other execution engines, such as Tez and Spark, which can enhance performance and usability. This will be discussed later.

Sub-Section: Comparing Hive to Traditional Databases

Comparison between Apache Hive and Traditional RDBMS (Part 1)

Feature	Apache Hive	Traditional RDBMS
Purpose	Big data analytics.	Transactional systems and traditional data warehousing.
Query Language	HiveQL, similar to SQL.	SQL.
Speed	Slower, optimized for batch processing.	Faster, optimized for real-time transactional processing.
Data Size	Designed to handle petabytes of data.	gigabytes to terabytes; some systems can handle petabytes at higher cost.
ACID Properties	Limited ACID support.	Full ACID support.

Table T-1: Comparison between Apache Hive and Traditional RDBMS

Comparison between Apache Hive and Traditional RDBMS (Part 2)

Feature	Apache Hive	Traditional RDBMS
Storage	Built on HDFS.	Uses internal storage mechanisms.
Metadata Storage	Stored in Hive Metastore.	Stored in system catalogs within the database.
Compute Engine	MapReduce, Tez, or Spark can be used.	Built-in engine, tightly integrated.
Query Execution Location	Executes on Hadoop cluster nodes.	Executes on the database server.
Data Storage Formats	ORC, Parquet, CSV, JSON, XML, Avro	Proprietary Binary Format
Max Simultaneous Connections	Governed by Hadoop Cluster	Varies (Hundreds to Thousands)

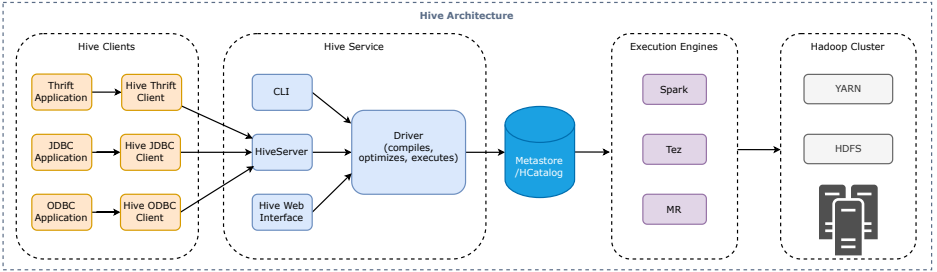
Table T-2: Comparison between Apache Hive and Traditional RDBMS

Section: Hive Architecture

Abstract Components of Apache Hive

- Hive Clients.
- Hive Services.
- Hive Metadata (Metastore).
- Storage.
- Compute.

Abstract Components of Apache Hive



Hive Clients

- Hive provides various drivers for seamless communication with different types of applications.
- For Thrift-based applications, Hive offers the Thrift client for effective communication.
- If you are working with Java-related applications, Hive provides JDBC drivers for smooth integration.
- Additionally, for other types of applications, Hive offers ODBC drivers, ensuring versatility.
- These clients and drivers serve as intermediaries, connecting your applications with Hive Server in the Hive services.

Hive Services

- Hive Services act as the intermediaries for client interactions with Hive.
- When clients need to perform query-related operations in Hive, they communicate through Hive Services.
- All drivers, including JDBC, ODBC, and other client-specific applications, communicate with Hive Server and the primary driver within Hive Services.
- The main driver in Hive Services processes requests from various applications, directing them to the metastore and data systems for further processing.

Hive Services | CLI (Continued)

- Hive Command Line Interface (CLI)
  - The CLI is the most common way to access Hive.
  - Its design can make it challenging to use programmatically.
  - It is a fat client, requiring a local copy of all Hive components and configurations.
  - It needs a copy of a Hadoop client and its configuration.
  - The CLI functions as an HDFS client, a MapReduce client, and a JDBC client (for accessing the metastore).
  - Even with the correct client installation, ensuring all necessary network access can be complex, especially across subnets or datacenters.

Hive Services | HiveServer2 (Continued)

- **HiveServer2**: HiveServer2 is a service that allows clients to submit HiveQL queries programmatically. It provides a remote interface for running Hive queries and managing sessions.
- **Thrift Service**: Hive uses the Apache Thrift framework to provide a cross-language service interface. This enables communication between clients and the Hive server.
- **Sessions**: When clients connect to HiveServer2, sessions are established to manage their interactions. Sessions help keep track of query state and context.

Hive Services | Hive Driver

- The Hive Driver is a critical component responsible for query execution.
- It consists of several key components:
  - Query Compiler.
  - Optimizer.
  - Execution
- Together, these components ensure efficient and effective query processing in Hive.

Hive Services | Hive Driver (Continued)

- Query Compiler
  - The Query Compiler takes HiveQL queries and translates them into executable jobs.
  - It's responsible for the logical and physical query planning, ensuring that the queries are optimized for efficient execution.

Hive Services | Hive Driver (Continued)

- Query Optimizer
  - It applies optimization techniques, including predicate pushdown and join optimization, to enhance query performance.
  - This ensures that queries are executed as efficiently as possible.

Hive Services | Hive Driver (Continued)

- The Execution Engine is responsible for the actual execution of queries.
- It encompasses several key tasks, including:
  - **Plan Execution**: It executes the query plan generated by the Query Compiler.
  - **Job(s) Generation**: Depending on the chosen execution engine (e.g., MapReduce), it generates the necessary jobs to process data in parallel.
  - **Submission to Hadoop**: It submits these jobs to the Hadoop cluster or other compatible compute environments for execution.
  - **Progress Monitoring**: It continuously monitors the progress of the query execution, providing insights into job completion and overall performance.

Metadata Store (e.g., MySQL)

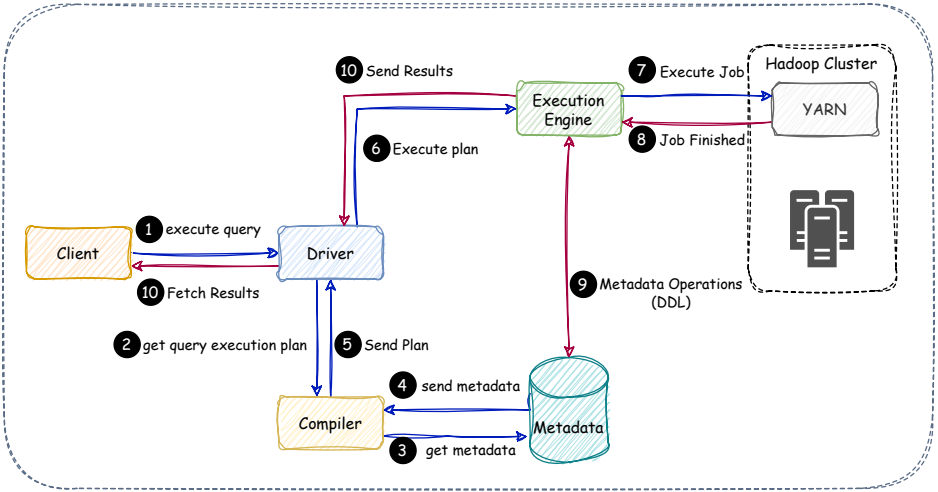
- The Metadata Store is a relational database, such as MySQL, that stores critical information about tables, columns, partitions, and their relationships.
- This database acts as a catalog, enabling Hive to understand the data's structure and schema.

Data Storage

- Hive operates on data stored in HDFS or compatible storage systems.
- Instead of transforming the data, it interprets it using a schema on read approach.
- This allows users to work with data without the need for extensive data preparation.

Sub-Section: Job Execution Flow in Hive

Job Execution Flow in Hive



Sub-Section: Hive Table Format

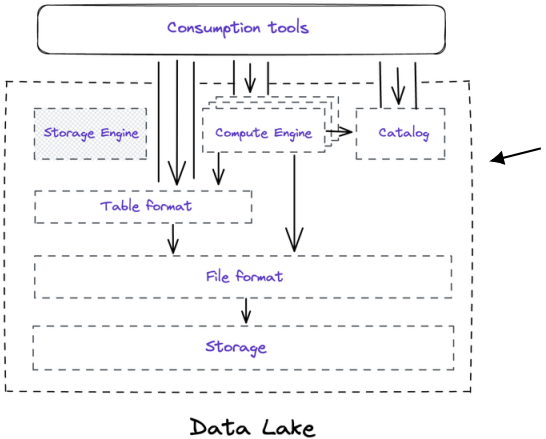
Hive Table Format

- Hive was created to convert SQL statements into MapReduce jobs.
- Mechanism needed for SQL to identify files and metadata.
- Hive was a breakthrough, linking directories and files to tables.
- HDFS directories map to database schemas and tables.

Hive Table Format | continued

- This was a breakthrough as it enabled working directly with the object store (i.e., HDFS) as the primary database storage, eliminating the need for custom file formats.
- It also introduced flexibility in adding more file input formats (open formats), such as CSV, ORC, AVRO, Parquet, etc. Any processing that can be done with Map-Reduce can be applied to Hive.
- Hive uses metastore/metadata to map raw HDFS data to named columns and types.
- Each Hive table belongs to a specific database.

Technical components in a data lake



Apache Iceberg: The Definitive Guide:
Data Lakehouse Functionality,
Performance, and Scalability
on the Data Lake
PUBLISHED BY: O'Reilly Media, Inc.

Figure F-1: Technical components in a data lake

Hive Table Format | Example of Text File

- Each Hive table links to a folder, usually on HDFS.
- This folder contains one or more text files.
- Basics of Hive's default text format:
  - One record by line (\n separator).
  - Columns are split by Control-A (^A).
  - Complex types use Control-B (^B).
  - Map keys and values use Control-C (^C).
- You can change these settings when making a new table.

Text File vs. Binary File in Hive | High Level

Criteria	Text File	Binary File (e.g., ORC, Parquet)
Readability	Human-readable	Not human-readable
Debugging	Easier	Harder
Storage Size	Larger	More efficient
Speed	Slower	Faster
Delimiters	Basic (e.g., Control-A)	Complex types supported
Suitability	Small datasets	Large datasets

Table T-3: Comparison between Text and Binary File Formats in Hive

Why Binary Format is Faster than Text Format

Factor	Text File	Binary File
Parsing	Needs conversion	Directly readable
Memory	Less efficient	Efficient
Storage	Larger files	Smaller due to compression
Compression	Basic	Advanced algorithms
IO Operations	More reads/writes	Fewer reads/writes
Schema Evolution	Harder	Easier

Table T-4: Summary: Binary formats are generally more efficient in reading, writing, and storing data, making them faster for large datasets.

Section: Hive Data Management

Sub-Section: Hive Database

Understanding Hive Database

- **What is Hive Database?**
  - A namespace for tables.
  - Helps in organizing data in Hive.
  - Namespaces function to avoid naming conflicts for tables, views, partitions, columns, and so on. Databases can also be used to enforce security for a user or group of users.
- **Table Organization**
  - Groups related tables under a single database.
  - Simplifies data management.
  - Homogeneous units of data which have the same schema.

Hive Warehouse Structure

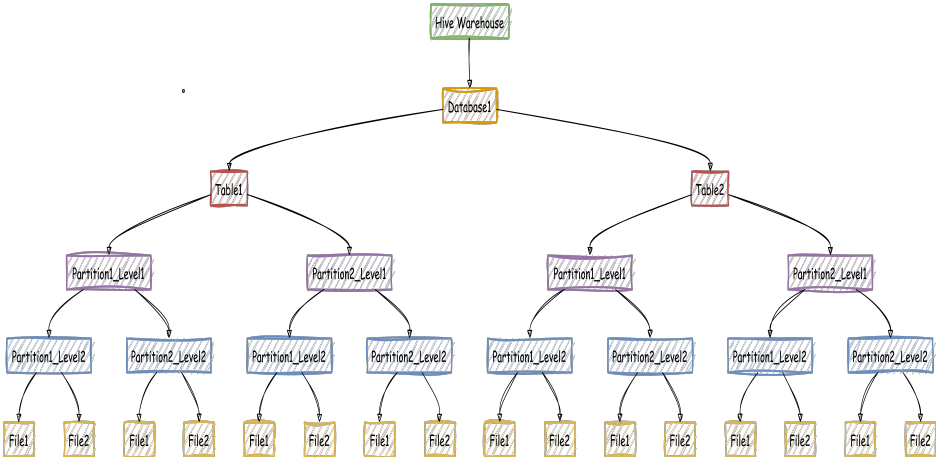


Figure F-2: Hive database structure

Hive Warehouse Structure | continued

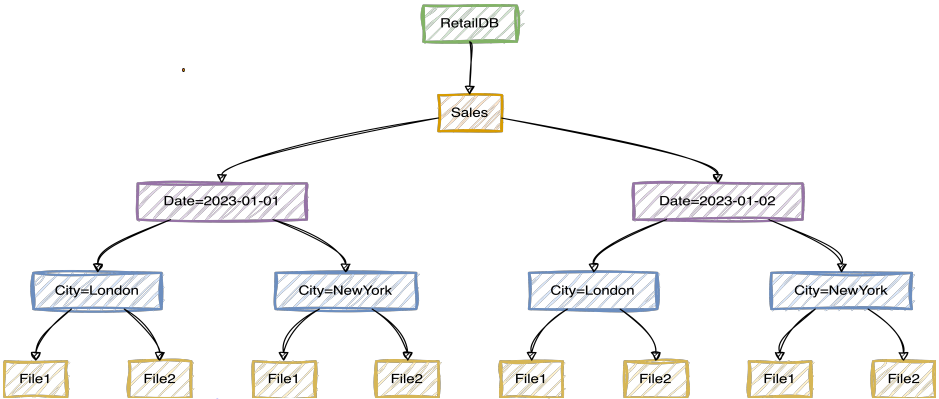


Figure F-3: Hive database structure | Retail DB Example

Hive Warehouse Structure | continued

Figure F-4: Hive database structure | Retail DB HDFS Structure

Creating Database in Hive

```
1 CREATE [REMOTE] (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
2 [COMMENT database_comment]
3 [LOCATION hdfs_path]
4 [MANAGEDLOCATION hdfs_path]
5 [WITH DBPROPERTIES (property_name=property_value, ...)];
```

Code Snippet: Create Database command in Hive

```
1 CREATE DATABASE hrDB;
2 USE hrDB;
```

Code Snippet: Create Database Example

Drop Database in Hive

```
1 DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

- Code Snippet: Drop Database command in Hive

Alter Database in Hive

```
1 ALTER (DATABASE|SCHEMA) database_name
2 SET DBPROPERTIES (property_name=property_value, ...);
```

Code Snippet: Alter Database command in Hive

```
1 ALTER (DATABASE|SCHEMA) database_name SET LOCATION hdfs_path;
```

Code Snippet: Alter Database Example

Use Database in Hive

```
1 USE database_name;
2 USE DEFAULT;
```

Code Snippet: Alter Database command in Hive

Hive Schema vs. Database

- In Hive, the terms 'schema' and 'database' are interchangeable.
- Both serve as a namespace for tables.

From Hive Documentation

"The uses of SCHEMA and DATABASE are interchangeable – they mean the same thing. CREATE DATABASE was added in Hive 0.6 (HIVE-675). The WITH DBPROPERTIES clause was added in Hive 0.7 (HIVE-1836)."

Sub-Section: Hive Tables

Creating Tables in Hive

```
1  CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
2  [( col_name data_type [column_constraint_specification] [COMMENT col_comment], ...
       [constraint_specification])]
3  [COMMENT table_comment]
4  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
5  [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)] INTO
       num_buckets BUCKETS]
6  [SKEWED BY (col_name, col_name, ...)
7  ON ((col_value, col_value, ...) , (col_value, col_value, ...) , ...)
8  [STORED AS DIRECTORIES]
9  [
10 [ROW FORMAT row_format]
11 [STORED AS file_format]
12 | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
13 ]
14 [LOCATION hdfs_path]
15 [TBLPROPERTIES (property_name=property_value, ...)]
16 [AS select_statement];
```

Code Snippet: Create Table Commands

CREATE TABLE in Hive | continued

Part 1: Table Creation Basics

- `CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name`
  - `TEMPORARY`: Creates a temporary table.
  - `EXTERNAL`: Defines the table as an external table.
  - `IF NOT EXISTS`: Only creates if the table does not exist.
  - `db_name.table_name`: Specifies database and table name.

CREATE TABLE in Hive | continued

Part 2: Column Definitions

- `[(col_name data_type [COMMENT col_comment], ...)]`
  - Define the columns and their data types.
  - Optional comment for each column.

CREATE TABLE in Hive | continued

Part 2: Column Definitions (data_type)

- **Primitive Type**: Basic data types like INT, BIGINT, BOOLEAN, FLOAT, DOUBLE, STRING, TIMESTAMP, DECIMAL, DATE, etc.
- **Array Type**: Ordered sequences of the same type, e.g., ARRAY<INT>.
- **Map Type**: Key-value pairs, e.g., MAP<STRING, INT>.
- **Struct Type**: Collection of named fields that can be different types, e.g., STRUCT<field1:STRING, field2:INT>.
- **Union Type**: Can be any of the specified types, e.g., UNIONTYPE<STRING, INT>.

CREATE TABLE in Hive | continued

Part 3: Row Format and SerDe in Hive

- `[ROW FORMAT row_format]`
  - Specifies the row format (e.g., delimited, sequence file, etc.)
  - Native SerDe is used if ROW FORMAT is not specified or ROW FORMAT DELIMITED is specified.
  - Use SERDE clause for custom SerDe.

CREATE TABLE in Hive | continued

RegEx Example

```
1 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
2 WITH SERDEPROPERTIES ("input.regex" = "<regex>")
3 STORED AS TEXTFILE;
```

Code Snippet: RegEx Example

CSV/TSV Example

```
1 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
2 WITH SERDEPROPERTIES ("separatorChar" = "\t","quoteChar" = """, "escapeChar" = "\\")
3 STORED AS TEXTFILE;
4 \end{verbatim}
```

Code Snippet: CSV/TSV Example

CREATE TABLE in Hive | continued

Part 4: File Format and Storage Handler

- `[STORED AS file_format | STORED BY 'storage.handler.class.name']`
  - Specifies the file format (e.g., TEXTFILE, PARQUET, SEQUENCEFILE, ORC, AVRO, JSONFILE etc.).
  - Custom storage handler (e.g., INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname)

CREATE TABLE in Hive | continued

Part 5: Table Partitions

- `[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]`
  - Specifies columns to be used for partitioning.
  - Partitioned tables can be created using the **`PARTITIONED BY`** clause.
  - A table can have one or more partition keys, and a separate data directory is created for each distinct value combination in the partition columns.
  - Partitioning enables efficient data filtering.

CREATE TABLE in Hive | continued

Part 5: Table Partitions

```
1  CREATE TABLE sales (
2  product_id INT,
3  order_date STRING,
4  amount DOUBLE
5  )
6  PARTITIONED BY (year INT, region STRING)
7  ROW FORMAT DELIMITED
8  FIELDS TERMINATED BY ','
9  STORED AS TEXTFILE;
```

Code Snippet: Create Partitioned Table

CREATE TABLE in Hive | continued

Part 5: Table Partitions

- **Table:** sales
- **Columns:** product_id, order_date, amount
- **Partition Columns:** year, region

This table stores sales records. Each record includes the *product_id*, *order_date*, and *amount* of the sale.

CREATE TABLE in Hive | continued

Part 5: Table Partitions

Figure F-5: Hive database structure | Retail DB HDFS Structure

CREATE TABLE in Hive | continued

Part 5: Table Partitions

```sql
SELECT * FROM sales WHERE year=2021 AND region='US';
```

Code Snippet: SQL

Non-Partitioned Table	Partitioned Table
Step 1: Full table scan	Step 1: Identify partitions (`year=2021, region='US'`)
Step 2: Apply WHERE filters (`year=2021 AND region='US'`)	Step 2: Scan only those partitions

Table T-5: Comparison: Non-Partitioned vs Partitioned Table.

CREATE TABLE in Hive | continued

Part 5: Table Partitions

```
1  EXPLAIN SELECT * FROM sales_non_partitioned WHERE year=2021 AND region='US';
```

Code Snippet: Explain Plan Command for Non-Partitioned Table

```
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: sales
            Filter Operator
              predicate: (year = 2021 and region = 'US')
```

Figure F-6: Non-Partitioned Table Execution Plan Summary

CREATE TABLE in Hive | continued

Part 5: Table Partitions

```
1  EXPLAIN SELECT * FROM sales_partitioned WHERE year=2021 AND region='US';
```

Code Snippet: Explain Plan Command for Partitioned Table

```
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: sales_partitioned
            Filter Operator
              predicate: (year = 2021 and region = 'US')
      Partition Pruning: (year = 2021 and region = 'US')
```

Figure F-7: Partitioned Table Execution Plan Summary

CREATE TABLE in Hive | continued

Part 5: Table Partitions

Aspect	Non-Partitioned	Partitioned
Full Table Scan	Yes	No
Partition Pruning	N/A	Yes
I/O Cost	High	Low
Time Complexity	More Time	Less Time
Resource Utilization	High	Low

Table T-6: Comparison: Non-Partitioned vs Partitioned Table.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting

- `[CLUSTERED BY (col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]`
  - Clustering and sorting options for better performance and to optimize data storage.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

ID	Name	Department	Salary
1	John	HR	5000
2	Alice	Sales	6000
3	Bob	IT	7000
4	Carol	HR	8000
5	Dave	Sales	9000
6	Eve	IT	4000

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

```
1  CREATE TABLE Employee (
2    ID INT,
3    Name STRING,
4    Department STRING,
5    Salary INT)
6  CLUSTERED BY (Department) INTO 3 BUCKETS;
```

Code Snippet: Create CLUSTERED Table

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

- Bucket 1: Data for HR
- Bucket 2: Data for Sales
- Bucket 3: Data for IT

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

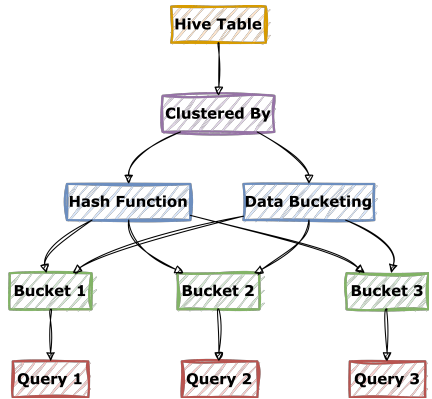


Figure F-8: Hive Table | Clustered by mechanism

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

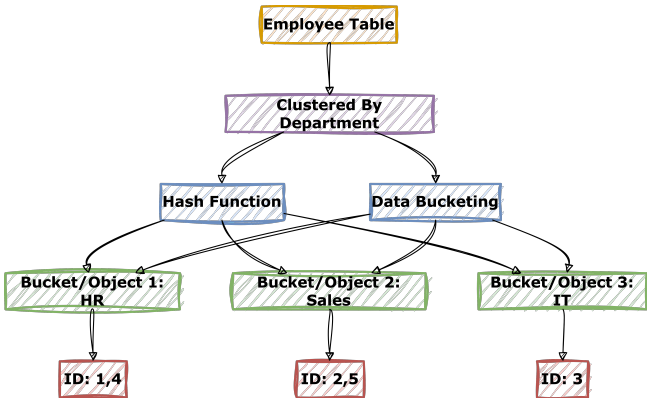


Figure F-9: Hive Table | Clustered by mechanism example

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

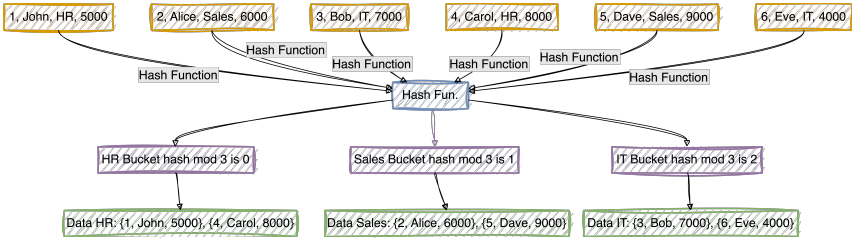


Figure F-10: Hive Table | Clustered by mechanism example

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

```
1  EXPLAIN SELECT * FROM Employee WHERE Department = 'HR';
```

Code Snippet: Simple SQL statement

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

```
ExplainPlan:
Stage:
  - Name: "Stage-1"
    Type: "Map Reduce"
  Operations:
    - TableScan:
      TableName: "Employee"
    - Filter :
      Condition: "Department='HR'"
```

Code Snippet: Simplified Explain Plan for Table Without CLUSTERED BY

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

```
ExplainPlan:
Stage: ^^I
  - Name: "Stage-1"
    Type: "Map Reduce"
    Operations:
      - TableScan:
          TableName: "Employee"
          Bucketing:
            PruningEnabled: true
            RelevantBuckets: "1/3"
      - Filter :
          Condition: "Department='HR'"
```

Code Snippet: Simplified Explain Plan for Table with CLUSTERED BY

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | CLUSTERED BY

- Faster JOIN operations.
- Optimized for grouped analysis.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | SORTED BY

```
1  CREATE TABLE Employee (
2    ID INT,
3    Name STRING,
4    Department STRING,
5    Salary INT)
6  CLUSTERED BY (Department) INTO 3 BUCKETS;
```

Code Snippet: With CLUSTERED BY and SORT BY

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | SORTED BY

```
1  CREATE TABLE Employee (
2    ID INT,
3    Name STRING,
4    Department STRING,
5    Salary INT)
6  CLUSTERED BY (Department) SORTED BY (ID) INTO 3 BUCKETS;
```

Code Snippet: With only CLUSTERED BY

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | SORTED BY

```
Query:
  Text: "EXPLAIN SELECT * FROM Employee WHERE Department = 'HR'"
  ExplainPlan:
  Stage:
    - Name: "Stage-1"
    Type: "Map Reduce"
    Operations:
      - TableScan:
        TableName: "Employee"
        Bucketing:
          PruningEnabled: true
          RelevantBuckets: "1/3"
      - Filter :
        Condition: "Department='HR'"
```

Code Snippet: Simplified Explain Plan With only CLUSTERED BY

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | SORTED BY

```
Query:
Text: "EXPLAIN SELECT * FROM Employee WHERE Department = 'HR'"
ExplainPlan:
  Stage:
   - Name: "Stage-1"
     Type: "Map Reduce"
     Operations:
      - TableScan:
        TableName: "Employee"
        Bucketing:
         PruningEnabled: true
         RelevantBuckets: "1/3"
      - Filter :
        Condition: "Department='HR'"
      - Sort:
        Columns: "ID"
```


CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | Clustered By: Key Points

- **Data Distribution**: Distributes rows based on hash value of one or more columns.
- **Query Optimization**: Faster responses when filtering by clustered columns.
- **Storage**: Organizes data in HDFS, improving data locality.
- **Better with Joins**: Joins on bucketed columns are optimized.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | Clustered By: When to Use

- **Large Datasets**: Effective for partitioning large datasets.
- **Common Queries**: Use when filtering or joining on specific columns.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | Sort By: Key Points

- **Ordering**: Sorts data within each bucket.
- **Local Sort**: Sorting is local to each bucket, not global.
- **Speed**: May speed up range-based queries within a bucket.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | Sort By: When to Use

- **Range Queries**: Useful for frequent range-based queries on a column within a bucket.
- **Ordered Reads**: Use if you need data in a specific order within each bucket.

CREATE TABLE in Hive | continued

Part 6: Clustering and Sorting | Sort By: When to Use

- Understand the query and consumption patterns.

CREATE TABLE in Hive | continued

Part 5: Data Skewing

- `[SKEWED BY (col_name, ...) ON ((col_value, ...), ...) [STORED AS DIRECTORIES]]`
  - Specifies skewed columns and values for better query performance.
  - Optionally store these as directories.

CREATE TABLE in Hive | continued

Part 8: Table Location

- `[LOCATION hdfs_path]`
  - Sets the HDFS directory where table data will be stored.

CREATE TABLE in Hive | continued

Part 9: Table Properties

- `[TBLPROPERTIES (property_name=property_value, ...)]`
  - Sets table-level properties.

CREATE TABLE in Hive | continued

Part 10: CTAS

- `[AS select_statement]`
  - Populates the table using the result set of a SELECT statement.

Sub-Section: Query Execution Plan

Query Execution Plan

- The Hive driver is responsible for translating SQL statements into an execution plan for the target execution engine.

- The process involves several key steps:
    1. The parser parses the SQL statement and generates an Abstract Syntax Tree (AST) representing logical operations like SELECTs, JOINs, UNIONs, groupings, and more.

    2. The planner retrieves table metadata from the Hive Metastore, including HDFS file locations, storage formats, row counts, etc.

    3. The query optimizer utilizes the AST and table metadata to produce a physical operation tree known as the execution plan, defining the physical operations needed to retrieve data, such as nested loop joins, sort-merge joins, hash joins, index joins, and more.

Query Execution Plan

- The execution plan determines the tasks executed on the Hadoop cluster and significantly impacts performance in data analytics systems like Hive.
- The execution plan generated by the query optimizer has a substantial impact on performance.
- Differences in the execution plan can result in significant variations in execution time, ranging from seconds to hours.
- An optimal execution plan is crucial for efficient query processing in Hive.

Query Execution Plan

- The Cost-Based Optimization (CBO) plays a pivotal role in enhancing the execution plan.
- CBO leverages table statistics to make informed decisions regarding the performance costs associated with each potential execution plan.
- This intelligent optimization ensures that the Hive driver produces an optimal execution plan, improving query performance.

Sub-Section: Cost-Based Optimization

Sub-Section: Hive Schema and Data Storage

Hive Schema and Data Storage

- Hive queries operate on tables, similar to RDBMS.
  - A table corresponds to a directory in storage (HDFS, S3, GCS, or Azure).
  - Each table comprises one or more files.
  - Every table is associated with a specific file format.
  - Hive stores table structure and location in the metadata store (RDBMS).
  - Hive supports various file formats, such as Parquet, ORC, and Text.

Hive Schema and Data Storage (Continued)

- Hive queries reference the metastore to access table location and structure.
- While queries interact with the file system, metadata is stored in the RDBMS.

Section: Further Readings and Assignment

Thank you for watching!

See you in the next video ☺