Unit - IV Hive and Pig

4.1 Introduction to Hive : Hive Characteristics, Limitations

Apache Hive is a data warehouse system built on top of Hadoop, used for querying and analyzing large datasets stored in HDFS using a SQL-like language called HiveQL

Characteristics:

- Provides **SQL-like language (HiveQL)**.
- Uses **Schema-on-Read** (flexible data loading).
- Designed for **batch processing**, not real-time.
- Supports many file formats (Text, ORC, Parquet, etc.).
- Stores metadata in **Metastore** (RDBMS).
- Highly **scalable** and **fault-tolerant** (inherits Hadoop features).
- Supports **UDFs** for custom logic.

Limitations of Hive

- **Not suitable for real-time** or low-latency queries.
- Weak for **row-level updates** (mainly append-only).
- Limited **ACID transaction** support.
- **Slow query execution** due to dependency on MapReduce/Tez/Spark.
- Not good for **OLTP** or frequent small reads/writes.
- Query optimization is weaker compared to RDBMS.

4.2 Hive Architecture

1. User Interface (UI)

- Provides tools like CLI, Hive Web UI, JDBC, ODBC to submit queries.

2. Driver

- Manages the lifecycle of a Hive query.
- Handles parsing, validation, session management.
- Coordinates with the compiler, optimizer, execution engine.

3. Compiler

- Converts HiveQL into an execution plan.
- Breaks query into DAG of MapReduce/Tez/Spark jobs.
- Performs logical optimization (pruning, transformation).

4. Metastore

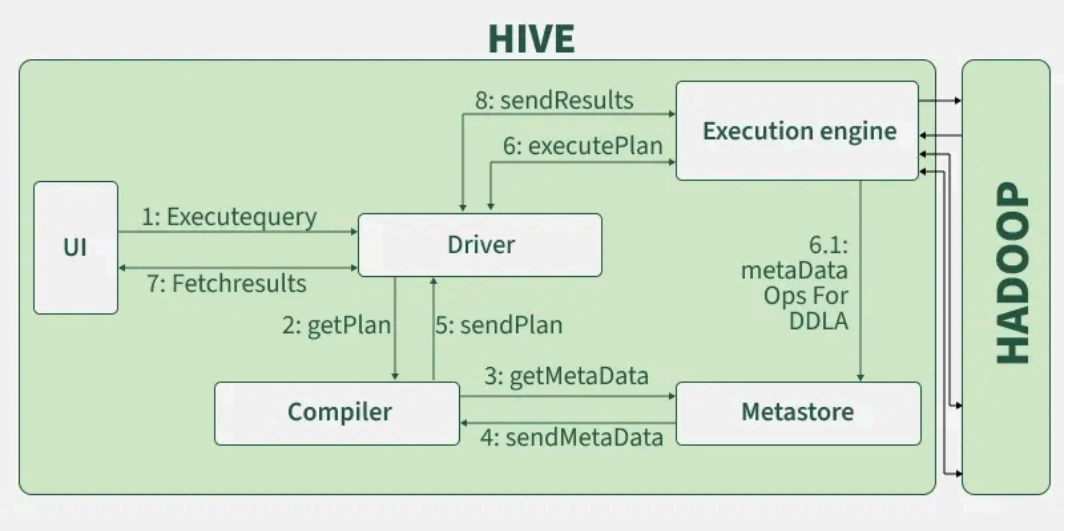
- Stores metadata (table schema, data types, partitions).
- Backed by RDBMS like MySQL/PostgreSQL.
- Essential for schema-on-read behavior.

5. Execution Engine

- Executes the optimized plan.
- Uses MapReduce, Tez, or Spark to run jobs.
- Interacts with Hadoop to read/write data from HDFS.

6. HDFS

- Actual storage layer for Hive tables.
- Stores data in formats like Text, ORC, Parquet, Avro, etc.



4.4 Workflow Steps

Working of Hive

1. **Query Execution:** User submits a query via CLI/Web UI/JDBC/ODBC; sent to the Driver.
2. **Plan Creation:** Driver sends query to Compiler to generate execution plan.

3. **Metadata Request:** Compiler requests required metadata from Metastore.
4. **Metadata Response:** Metastore sends metadata back to Compiler.
5. **Plan Submission:** Compiler returns optimized plan to **Driver**.
6. **Execute Plan:** Execution Engine runs plan using MapReduce/Tez/Spark.
7. **Return Results:** Final output sent back to the user through the **UI**.

4.3 Hive Data Types and File Formats

Hive Data Types

1. **Primitive Types**

- **Numeric:** TINYINT, INT, BIGINT, FLOAT, DOUBLE, DECIMAL
- **String:** STRING, CHAR, VARCHAR
- **Boolean:** BOOLEAN
- **Date/Time:** DATE, TIMESTAMP
- **Binary:** BINARY

2. **Complex Types**

- **ARRAY** – list of values
- **MAP** – key–value pairs
- **STRUCT** – record with multiple fields
- **UNIONTYPE** – one of several types

Hive File Formats

- **Text** – simple, readable, low efficiency
- **SequenceFile** – binary key-value, faster than text
- **RCFile** – early columnar format
- **ORC** – optimized columnar, high compression, best for analytics
- **Parquet** – modern columnar format, great with Spark
- **Avro** – row-based, good for data exchange and schema evolution

4.4 Hive Integration

1. **Hadoop Integration:**

   Hive works on top of **HDFS** and uses **MapReduce/Tez/Spark** for processing.

2. **Metastore Integration:**

   Uses external **RDBMS** (MySQL/PostgreSQL) to store table metadata.

3. **BI / Analytics Tools:**

   Tools like **Tableau, Power BI, Qlik** connect via **JDBC/ODBC**.

4. **Programming Integration:**

   Applications can access Hive using **JDBC/ODBC drivers** or **HiveServer2**.

5. **Ecosystem Integration:**

   Works with **Sqoop, Flume**, **HBase**, and **Oozie** for ETL, data ingestion, and workflow scheduling.

4.5 Hive Built-in functions

Category	Function	Description
String Functions	LOWER(str) / UPPER(str)	Convert case
	LENGTH(str)	Returns string length
	TRIM(str)	Removes leading/trailing spaces
	SUBSTR(str, start, len)	Extract substring
	CONCAT(a, b)	Join strings
	REVERSE(str)	Reverse text
Date & Time Functions	CURRENT_DATE()	Returns current date
	CURRENT_TIMESTAMP()	Returns current timestamp
	DATE_ADD(date, n)	Add days
	DATE_SUB(date, n)	Subtract days

	YEAR(date), MONTH(date), DAY(date)	Extract date parts
	UNIX_TIMESTAMP()	Date → UNIX time
Math / Numeric Functions	ABS(x)	Absolute value
	ROUND(x)	Round number
	CEIL(x) / FLOOR(x)	Upper/lower integer
	SQRT(x)	Square root
	RAND()	Random number
Conditional Functions	IF(cond, x, y)	If-else condition
	COALESCE(a, b, ...)	First non-null value
	CASE WHEN ... THEN ... END	Conditional logic
Aggregate Functions	COUNT()	Count rows
	SUM(), AVG()	Sum, average
	MIN(), MAX()	Minimum, maximum
Table-Generating Functions	EXPLODE(arr/map)	Creates multiple rows
	POSEXPLODE()	Explode with position
Collection Functions	SIZE(arr/map)	Size of array/map
	MAP_KEYS(map)	Extract keys
	MAP_VALUES(map)	Extract values

4.6 HiveQL : HiveQL DDL, HiveQL DML, HiveQL for Querying the Data

HiveQL is a SQL-like language used in Hive for defining, manipulating, and querying data stored in HDFS.

HiveQL DDL (Data Definition Language)

Used to **create**, **alter**, and **drop** databases or tables.

Command	Purpose (Short)	Example

CREATE DATABASE	Create a new database	CREATE DATABASE sales;
CREATE TABLE	Create table with schema	CREATE TABLE emp(id INT, name STRING);
EXTERNAL TABLE	Table storing data outside Hive	CREATE EXTERNAL TABLE ...
PARTITIONED TABLE	Table with partitioning	PARTITIONED BY (year INT)
ALTER TABLE	Modify table (add/rename columns/partitions)	ALTER TABLE emp ADD COLUMNS (age INT);
DROP TABLE	Delete table and metadata	DROP TABLE emp;
SHOW TABLES	List tables	SHOW TABLES;

HiveQL DML (Data Manipulation Language)

Used to **insert**, **update**, **delete**, and **load** data.

Command	Purpose (Short)	Example
LOAD DATA	Load file into Hive table	LOAD DATA INPATH '/data/emp' INTO TABLE emp;
INSERT INTO	Insert records	INSERT INTO emp VALUES (1,'John');
INSERT OVERWRITE	Replace existing data	INSERT OVERWRITE TABLE emp SELECT * FROM new_emp;
UPDATE	Update rows (ACID tables only)	UPDATE emp SET age=30 WHERE id=1;
DELETE	Delete rows (ACID tables only)	DELETE FROM emp WHERE id=1;

HiveQL for Querying the Data

Used to **retrieve and analyze** stored data.

Query Type	Purpose (Short)	Example

SELECT	Retrieve data	SELECT * FROM emp;
WHERE	Filter rows	SELECT * FROM emp WHERE age>25;
GROUP BY	Group rows for aggregation	SELECT dept, COUNT(*) FROM emp GROUP BY dept;
ORDER BY	Full sort	SELECT * FROM emp ORDER BY age;
SORT BY	Sort per reducer (partial)	SELECT * FROM emp SORT BY age;
DISTRIBUTE BY	Control reducer distribution	DISTRIBUTE BY dept;
JOIN	Combine tables	SELECT e.name, d.dept FROM emp e JOIN dept d ON e.dept_id=d.id;
LIMIT	Limit number of output rows	SELECT * FROM emp LIMIT 10;

4.7 Introduction to Pig : Applications of Apache Pig, Features of Pig, Compare Pig with SQL, MapReduce, and Hive

Apache Pig is a high-level platform for processing large data sets on Hadoop. Uses Pig Latin, a data flow language, to transform and analyze data. Simplifies complex MapReduce tasks with a script-based approach.

Applications of Apache Pig

1. **ETL (Extract, Transform, Load)** pipelines.
2. **Data preprocessing** for analytics.
3. **Log analysis** (web logs, server logs).
4. **Ad-hoc data analysis**.
5. **Iterative processing** (for machine learning or graph analysis).

Features of Pig

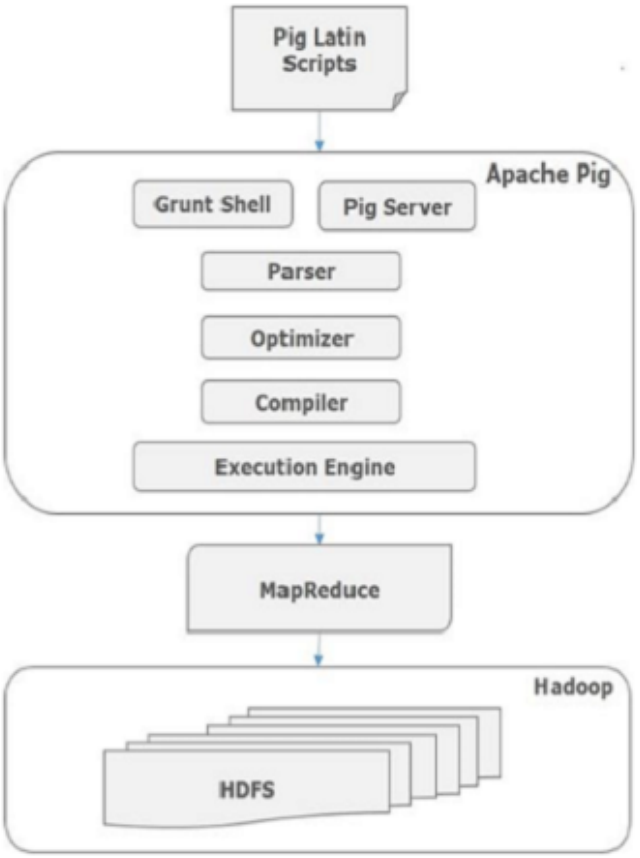
- High-level scripting language (Pig Latin).
- Extensible with UDFs (User Defined Functions).
- Supports structured, semi-structured, and unstructured data.
- Automatic optimization of execution plans.

- Handles large datasets efficiently on Hadoop.
- Schema-on-read capability.

Compare Pig with SQL, MapReduce, and Hive

Feature	Pig	SQL	MapReduce	Hive
Language Type	Pig Latin (procedural)	SQL (declarative)	Java/Custom code	HiveQL (declarative)
Ease of Use	Easier than MapReduce	Very easy	Hard (requires coding)	Easy
Data Handling	Structured & unstructured	Structured	Structured & unstructured	Structured
Optimization	Automatic execution plan	Query optimizer	Manual	Some automatic optimization
Performance	Faster for pipelines than Hive sometimes	Depends on DB engine	Efficient but complex	Depends on execution engine
Extensibility	UDF support	Limited	Full (Java)	UDF support

4.8 Pig Architecture



- **Pig Latin Scripts (User Layer)**

  Users write **Pig Latin scripts** to perform data transformations and analysis.

- **Apache Pig Components**
  1. **Grunt Shell** – Command-line shell to write and run Pig Latin scripts.
  2. **Pig Server** – API/interface to submit Pig scripts.
  3. **Parser** – Checks syntax of Pig Latin scripts and creates a plan.
  4. **Optimizer** – Improves the logical plan for better performance.
  5. **Compiler** – Converts the optimized logical plan into a physical plan.
  6. **Execution Engine** – Executes the physical plan on **Hadoop cluster** and returns results.

- **MapReduce** – Processing frameworks that execute Pig's physical plan jobs.

- **Hadoop**
  1. **HDFS** – Distributed storage system where Pig reads/writes data.

4.9 Pig Latin Data Model

Pig works with a **nested, complex data model** that supports **structured, semi-structured, and unstructured data**.

1. Atom

- **Atomic value** (primitive data).
- Examples: int, float, chararray, bytearray.

2. Tuple

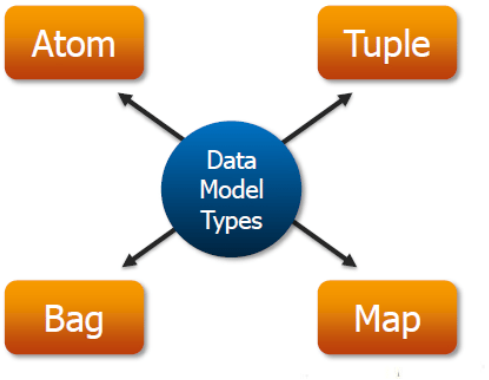
- **Ordered set of fields**, similar to a row in a table.
- Example: (1, 'John', 25)

3. Bag

- **Collection of tuples** (unordered, can have duplicates).
- Example: {(1,'John'), (2,'Alice')}

4. Map

- **Key-value pairs** within a tuple or bag.
- Example: ['name'#'John', 'age'#25]



Questions:

1. Describe Hive Query Language.
2. What is Hive ? Describe Hive Architecture.
3. List all hive data types.
4. Describe Hive file format.
5. Sketch Hive architecture.
6. Execute various commands to create a Hive table.
7. State the use of HIVE.
8. Explain HIVE Data Types.
9. Write syntax and example of HIVE Query Commands for following :
   (1) Create Table (2) Alter Table
10. Describe HIVE Architecture
11. Explain HIVE file format.