Apache Hive

NIELIT Chandigarh/Ropar

"Big Data is at the foundation of all the megatrends that are happening today."

Apache Hive

Introduction:

- Apache Hive is a **data warehouse** system built on top of **Hadoop**. It helps you to query and manage large amounts of data stored in Hadoop's **HDFS** (Hadoop Distributed File System) using a **SQL-like language** called **HiveQL**. It makes big data more accessible to people who are familiar with SQL, rather than requiring knowledge of programming languages like Java.
- **Data Warehousing Tool for Hadoop:** Apache Hive is a powerful data warehousing solution designed to work on top of the Hadoop framework. It enables users to query and analyze large datasets stored in Hadoop using a familiar SQL-like interface.

HiveQL:

- **SQL-like Query Language for Big Data:** Hive provides a querying language called HiveQL, which closely resembles SQL. This makes it accessible for users already familiar with SQL to interact with data stored in Hadoop's distributed file system.

Who Developed Apache Hive?

- Apache Hive was originally developed by Facebook in 2007.
- It was created to make querying and analyzing large amounts of data easier.
- Facebook engineers needed a way to analyze massive datasets using SQL queries, but Hadoop's native tools were too complex for this task.
- In 2008, Facebook contributed Hive to the Apache Software Foundation, where it became an open-source project.

Why Was Apache Hive Developed?

- The development of Hive was driven by the need to process huge amounts of data stored on Hadoop. Some key reasons for developing Hive:

1. **Ease of Use**:
   1. Traditional **SQL** is easy to understand and widely used. Hive was created to provide an easy way for people familiar with SQL to interact with Hadoop, without having to learn complex MapReduce programming.
2. **Handling Big Data**:
   1. As companies like Facebook started generating enormous amounts of data, they needed a tool to manage and query it efficiently. Hive helps in storing, querying, and analyzing massive datasets.
3. **Batch Processing**:
   1. Hive was designed for **batch processing**, meaning it's ideal for analyzing large volumes of data over time (like daily reports or trend analysis). It is not built for real-time processing.
4. **SQL-Like Language (HiveQL)**:
   1. Hive provides a **SQL-like query language** called HiveQL that's similar to traditional databases. This makes it easy for developers who are familiar with SQL to use it without needing to know Hadoop-specific programming languages like Java.
5. **Scalability**:
   1. Hive allows businesses to scale their data processing needs with Hadoop's **distributed storage** and **processing capabilities**, making it suitable for very large datasets.

In Simple Terms:

- Hive is like a **bridge** between the traditional world of SQL databases and the **big data world** powered by Hadoop.
- It was developed to make working with huge datasets more **manageable and understandable** for people who know SQL.

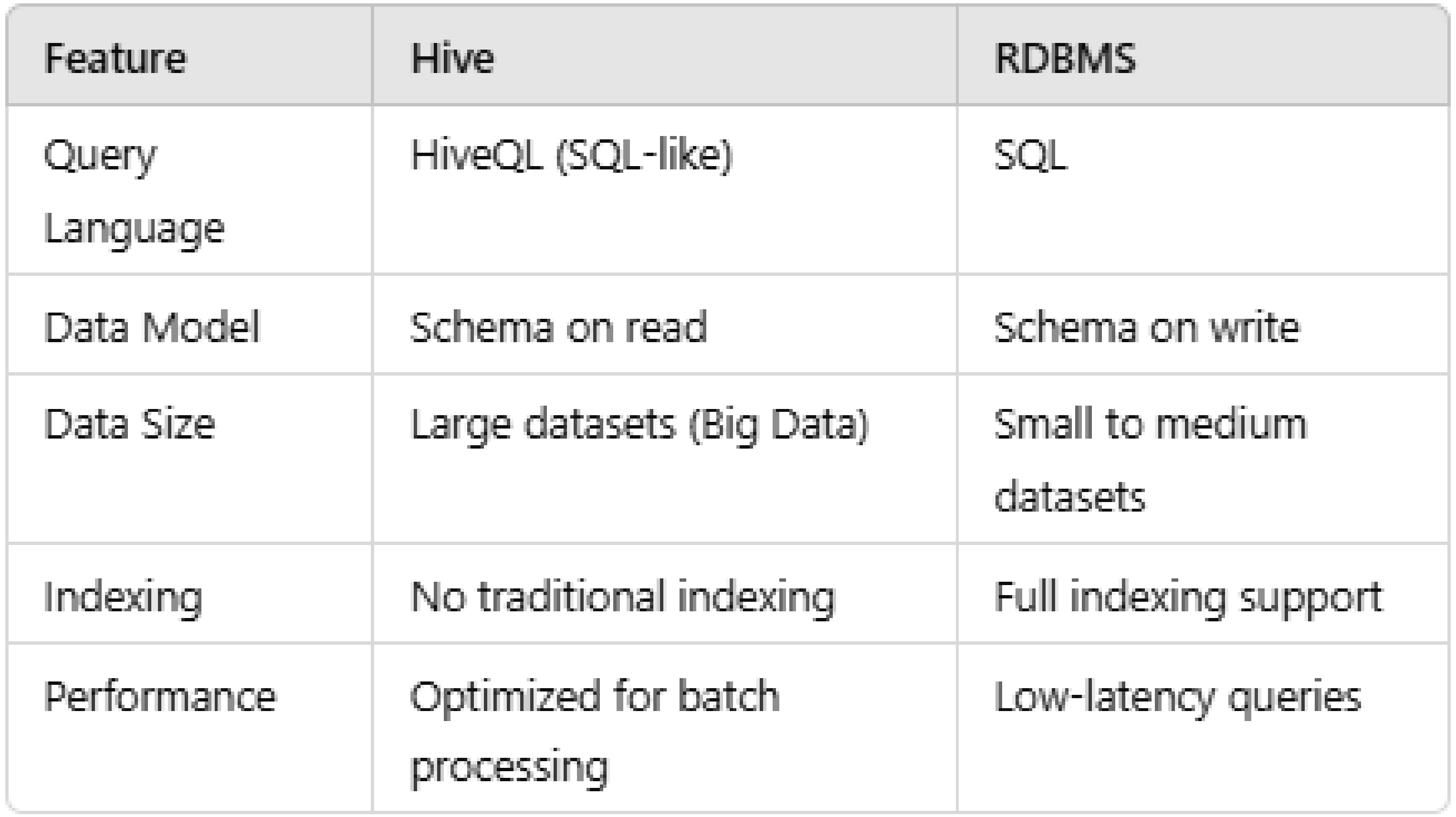
Key Features

- **SQL-Like Query Language**: Hive Query Language (HQL) similar to SQL.
- **Scalable and Distributed**: Built on Hadoop, it scales with your data.
- **Extensibility**: Users can add custom functions (UDFs).
- **Batch Processing**: Ideal for batch processing and not for low-latency operations.
- **Integration with Hadoop Ecosystem**: Can be used alongside Hadoop tools like HDFS, YARN, and MapReduce.

Hive Architecture

- **Components**:**Hive Metastore**: Stores metadata like schema, tables, partitions.
- **Driver**: Processes the queries.
- **Compiler**: Converts queries to a directed acyclic graph (DAG) for execution.
- **Execution Engine**: Executes the query plan, usually using MapReduce.
- **Hadoop/HDFS**: Underlying storage and processing layer.

Hive vs. Traditional RDBMS



Feature	Hive	RDBMS
Query Language	HiveQL (SQL-like)	SQL
Data Model	Schema on read	Schema on write
Data Size	Large datasets (Big Data)	Small to medium datasets
Indexing	No traditional indexing	Full indexing support
Performance	Optimized for batch processing	Low-latency queries

Advantages

User-Friendly Interface:

- Hive’s SQL-like syntax makes it easy for traditional database developers and analysts to adopt without a steep learning curve.

Quick Insights on Large Datasets:

- Hive is optimized for batch processing of large datasets, providing fast and scalable solutions for generating insights from big data.

Advantages

- Scalability: Apache Hive is designed to handle large volumes of data, making it a scalable solution for big data processing.
- Familiar SQL-like interface: Hive uses a SQL-like language called HiveQL, which makes it easy for SQL users to learn and use.
- Integration with Hadoop ecosystem: Hive integrates well with the Hadoop ecosystem, enabling users to process data using other Hadoop tools like Pig, MapReduce, and Spark.
- Supports partitioning and bucketing: Hive supports partitioning and bucketing, which can improve query performance by limiting the amount of data scanned.
- User-defined functions: Hive allows users to define their own functions, which can be used in HiveQL queries.

Disadvantages

- Limited real-time processing: Hive is designed for batch processing, which means it may not be the best tool for real-time data processing.
- Slow performance: Hive can be slower than traditional relational databases because it is built on top of Hadoop, which is optimized for batch processing rather than interactive querying.
- Steep learning curve: While Hive uses a SQL-like language, it still requires users to have knowledge of Hadoop and distributed computing, which can make it difficult for beginners to use.
- Limited flexibility: Hive is not as flexible as other data warehousing tools because it is designed to work specifically with Hadoop, which can limit its usability in other environments.

Hive Data Types

- **Primitive Data Types**: INT, BIGINT, STRING, BOOLEAN, FLOAT, DOUBLE, TIMESTAMP
- **Complex Data Types**:
  - **ARRAY**: An ordered collection of elements.
  - **MAP**: A key-value pair.
  - **STRUCT**: A collection of fields (like a record).
  - **UNIONTYPE**: A choice of different types.

Creating Tables in Hive

Create Table:

```
CREATE TABLE employees (
   id INT,
   name STRING,
   age INT
);
```

Partitioned Tables:

```
CREATE TABLE sales (
   id INT,
   amount DOUBLE,
   product STRING
)
PARTITIONED BY (year INT, month INT);
```

When you load data into a partitioned table, you specify the partition values:

```
LOAD DATA INPATH '/path/to/sales_data' INTO TABLE sales PARTITION (year=2022, month=01);
```

Querying the table by specifying partition columns can help reduce the amount of data scanned:

```
SELECT * FROM sales WHERE year=2022 AND month=01;
```

Hive will only scan the partition corresponding to the year 2022 and month 01, rather than scanning the entire table.

Querying Data in Hive

- Select Query:

```
SELECT * FROM employees;
```

- Filter Data:

```
SELECT name, age FROM employees WHERE age > 30;
```

- Joins:

```
SELECT a.id, a.name, b.department
FROM employees a
JOIN departments b ON a.department_id = b.id;
```

Loading Data into Hive

- Loading Data from HDFS:

```
LOAD DATA INPATH '/path/to/data' INTO TABLE employees;
```

- External Tables: Allows loading data from external sources without moving it into Hive’s internal storage.

```
CREATE EXTERNAL TABLE employees (
    id INT,
    name STRING,
    age INT
)
STORED AS TEXTFILE
LOCATION '/path/to/external/data';
```

An **external table** in Hive refers to a table where the data is stored **outside of Hive's managed storage** (typically HDFS or another filesystem like S3). The table in Hive doesn't control the underlying data; it merely references the external data files.

Hive Query Execution Flow

1. **HiveQL**: The user submits a HiveQL query.
2. **Parsing**: The query is parsed and converted into a query execution plan.
3. **Compilation**: The query is compiled into MapReduce jobs.
4. **Execution**: The execution engine runs the MapReduce jobs on Hadoop clusters.
5. **Results**: The result is returned to the user.

Performance Optimization in Hive

- **Partitioning**: Dividing data into partitions to reduce the amount of data scanned.
- **Bucketing**: Grouping data into buckets for efficient sampling.
- **Indexes**: Hive supports limited indexing features to speed up certain queries.
- **Query Optimization**: Partition pruning, vectorized execution, etc.

Hive Limitations

- **No Support for Real-time Data**: Hive is designed for batch processing, not for real-time analytics.
- **Limited Indexing**: Unlike traditional RDBMS, Hive doesn't support traditional indexes.
- **No Full ACID Compliance**: It is evolving to support limited ACID operations but not at the level of RDBMS systems.
- **Higher Latency**: Queries can have higher latency due to reliance on MapReduce.

Hive Use Cases

- **Data Warehousing**: Hive is typically used for data warehousing and large-scale analytics.
- **Log Analysis**: Processing and analyzing log data from various sources.
- **Data Transformation**: Performing ETL (Extract, Transform, Load) operations on large datasets.
- **Batch Analytics**: Running long-running batch queries to analyze historical data.

Integrating Hive with Other Tools

- **Apache HBase**: Hive can integrate with HBase for real-time querying.
- **Apache Spark**: Hive can be used as a source and sink for Spark data processing.
- **Apache Impala**: Can be used as an interactive query engine for Hive data, offering faster query performance than Hive on MapReduce.

Conclusion

- **Summary**: Apache Hive is a powerful tool for processing large-scale datasets using SQL-like queries, optimized for Hadoop’s distributed storage and batch processing.
- **Future**: With the evolution of newer tools like Apache Spark and Apache Impala, Hive’s role in real-time querying is growing, though it remains a cornerstone of big data analytics.

Apache Hive Setup and Project Creation (Step-by-Step Guide)