Hive interview Questions

1. What is the definition of Hive? What is the present version of Hive?

Hive is an open-source data warehouse infrastructure and query execution framework built on top of Apache Hadoop. It provides a SQL-like language called HiveQL (HQL) for querying and analyzing large datasets stored in distributed storage systems such as Hadoop Distributed File System (HDFS) or Apache HBase. Hive allows users to define schemas and tables, write queries using HiveQL, and perform data analysis tasks in a distributed computing environment. It provides tools and utilities for managing and processing structured and semi-structured data, making it easier for users familiar with SQL to work with big data.

The present version of hive is 4.0.0 alpha 2.

1. Is Hive suitable to be used for OLTP systems? Why?

No, Hive is not suitable for OLTP (Online Transaction Processing) systems. Hive is primarily designed for OLAP (Online Analytical Processing) workloads and is optimized for performing complex analytical queries on large datasets.

Here are some reasons why Hive is not suitable for OLTP systems:

1. Latency: Hive is optimized for batch processing and is not designed for low-latency operations. It typically operates on large volumes of data and focuses on throughput rather than quick response times. OLTP systems, on the other hand, require fast response times for handling real-time transactions.

2. Data Updates: Hive is designed for write-once, read-many (WORM) scenarios, where data is typically loaded in bulk and not frequently updated or modified. OLTP systems, on the other hand, involve frequent data updates, inserts, and deletes, which is not the primary focus of Hive.

3. Data Modeling: Hive uses a columnar storage format and is based on a schema-on-read approach, allowing for flexible and dynamic data modeling. OLTP systems, on the other hand, often require a strict schema and rely on a schema-on-write approach for data consistency and integrity.

4. ACID Transactions: Hive does not provide built-in support for ACID (Atomicity, Consistency, Isolation, Durability) transactions, which are essential for maintaining data integrity and consistency in OLTP systems.

1. How is HIVE different from RDBMS? Does hive support ACID transactions. If not then give the proper reason.

Hive is different from a traditional Relational DatabaseManagement System (RDBMS) in several ways:

1. Data Model: Hive follows a schema-on-read approach, where the schema is defined at the time of querying the data. It allows flexibility in handling different data formats and structures, making it suitable for handling semi-structured and unstructured data. RDBMS, on the other hand, follows a schema-on-write approach, where the schema is predefined and enforced during data insertion.

2. Query Language: Hive uses HiveQL (HQL), which is a SQL-like language specifically designed for querying and analyzing data stored in distributed systems like Hadoop. It provides a familiar SQL syntax and supports various data transformations and analytical functions. RDBMS, such as MySQL or PostgreSQL, uses SQL as its query language.

3. Scalability: Hive is designed to handle large-scale data processing on distributed systems like Hadoop. It leverages the scalability and fault-tolerance features of the underlying infrastructure. RDBMS, while scalable, is generally not built for handling massive datasets and distributed processing.

4. Data Storage: Hive is often used with distributed storage systems like Hadoop Distributed File System (HDFS) or Apache HBase. It leverages the distributed storage capabilities for handling big data. RDBMS typically uses local disk storage or network-attached storage (NAS) for data storage.

Regarding ACID transactions, Hive does not provide built-in support for ACID transactions. ACID (Atomicity, Consistency, Isolation, Durability) transactions ensure data integrity and consistency in a database system. Hive focuses on processing large-scale batch data analytics and does not prioritize real-time transactional processing.

The absence of ACID transactions in Hive is primarily due to its design goals and focus on scalability and throughput. Hive is optimized for query performance on large datasets, and providing full ACID transaction support would introduce additional overhead and complexity, which could impact its analytical processing capabilities.

However, it's worth noting that there are efforts and projects in the Hadoop ecosystem, such as Apache HBase and Apache Phoenix, that provide ACID transaction support and can be integrated with Hive for specific use cases requiring transactional capabilities.

1. Explain the hive architecture and the different components of a Hive architecture?

The Hive architecture consists of different components that work together to enable data processing and analysis on large-scale datasets. Here are the key components of the Hive architecture:

1. Hive Clients: Hive provides various client interfaces to interact with the Hive system. These clients include the Hive Command Line Interface (CLI), Hive Shell, Hive Web Interface, and various programming language APIs such as JDBC and ODBC. Clients are used to submit queries, manage metadata, and retrieve query results.

2. User Interface: The User Interface (UI) component provides a graphical or command-line interface for users to interact with Hive. It allows users to submit queries, monitor query progress, and view query results.

3. Hive Driver: The Hive Driver is responsible for receiving queries from the clients and initiating the query execution process. It parses the query, performs query optimization, generates an execution plan, and coordinates the execution across the other components.

4. Hive Metastore: The Hive Metastore is a central component that stores and manages metadata information related to Hive tables, partitions, columns, and other objects. It stores this metadata in a persistent database such as Apache Derby, MySQL, or PostgreSQL. The Metastore provides schema information, table statistics, and other details necessary for query execution.

5. Query Compiler and Optimizer: The Query Compiler and Optimizer component takes the query generated by the Hive Driver and performs optimization techniques to enhance query performance. It applies various optimizations such as predicate pushdown, join reordering, and column pruning to improve query execution efficiency.

6. Execution Engine: The Execution Engine is responsible for executing the query plan generated by the Query Compiler and Optimizer. Hive supports multiple execution engines, including MapReduce, Tez, and Spark. These engines leverage the underlying distributed processing framework to execute queries in parallel across multiple nodes.

7. Hive Warehouse: The Hive Warehouse is a storage location where Hive data is stored. It can be a distributed file system like Hadoop Distributed File System (HDFS) or a cloud-based storage system like Amazon S3. The Hive Warehouse contains the actual data files that are queried and analyzed by Hive.

8. External Storage Systems: Hive integrates with external storage systems like HBase, Apache Kafka, or relational databases through connectors and storage handlers. This allows Hive to query data residing in these external systems and combine it with data stored in the Hive Warehouse.

9. SerDe: SerDe (Serializer/Deserializer) is a crucial component that handles the serialization and deserialization of data between Hive tables and the underlying storage format. SerDes define how data is serialized when inserted into Hive tables and deserialized when queried or retrieved.

These components work together to provide a comprehensive architecture for data processing and analysis using Hive. They enable the management of metadata, query execution, optimization, and integration with various storage systems, making it easier to work with large-scale datasets in a SQL-like environment.

1. Mention what Hive query processor does? And Mention what are the components of a Hive query processor?

The Hive query processor is responsible for processing HiveQL queries and converting them into an optimized execution plan. It performs various tasks such as query parsing, semantic analysis, query optimization, and generating an execution plan for query execution.

The components of a Hive query processor include:

1. Parser: The Parser component takes the input HiveQL query and converts it into an abstract syntax tree (AST). It ensures the query syntax is correct and validates it against the grammar rules defined by the HiveQL language.

2. Semantic Analyzer: The Semantic Analyzer performs semantic analysis on the AST generated by the parser. It checks for semantic errors, resolves table and column references, verifies data types, and enforces semantic rules defined by Hive.

3. Query Optimizer: The Query Optimizer component applies various optimization techniques to improve query performance. It analyzes the query and identifies optimization opportunities such as predicate pushdown, join reordering, and column pruning. The optimizer rewrites the query plan to execute it more efficiently.

4. Cost-Based Optimizer: The Cost-Based Optimizer, also known as the CBO, is an optional component in the Hive query processor. It takes into account statistical information about tables, partitions, and column data to estimate the cost of different query plans. The CBO selects the most optimal plan based on cost estimates.

5. Physical Plan Generator: The Physical Plan Generator takes the optimized logical query plan and generates a physical query plan suitable for execution. It determines the most efficient way to execute the query based on the chosen execution engine, such as MapReduce, Tez, or Spark.

6. Query Executor: The Query Executor component executes the physical query plan generated by the Physical Plan Generator. It coordinates the execution across the cluster, manages task scheduling, data movement, and ensures fault tolerance and data consistency.

7. Result Set Fetcher: The Result Set Fetcher retrieves the query results from the execution engine and returns them to the user. It handles the retrieval, formatting, and presentation of the query results, making them available to the client application or user interface.

These components work together to process HiveQL queries, validate their syntax and semantics, optimize the query plan, and execute the query using the chosen execution engine. The query processor plays a crucial role in enabling efficient and effective data processing and analysis in Hive.

1. What are the three different modes in which we can operate Hive?

Hive can be operated in three different modes:

.

1. Local Mode: In Local Mode, Hive operates in a standalone manner on the local machine without connecting to a distributed cluster. It uses the local file system for data storage and processing. This mode is suitable for small-scale data processing and development purposes when working with relatively small datasets.

2. MapReduce Mode: In MapReduce Mode, Hive leverages the Hadoop MapReduce framework for distributed data processing. It submits MapReduce jobs to a Hadoop cluster for query execution. Hive takes advantage of the scalability and fault tolerance provided by Hadoop's distributed processing capabilities. This mode is suitable for handling large-scale data processing and analysis.

3. Tez Mode: Tez Mode is an alternative execution mode in Hive that utilizes the Apache Tez framework for query execution. Tez is a high-performance data processing engine that offers improved performance compared to MapReduce. It provides a more efficient execution environment for complex queries involving multiple stages and joins. Tez Mode is especially beneficial for interactive and ad-hoc query processing

Both MapReduce Mode and Tez Mode operate in a distributed environment and can handle big data workloads. The choice between these modes depends on factors such as query complexity, performance requirements, and the underlying infrastructure available. The mode can be specified during Hive configuration or while executing queries using appropriate commands or settings.

1. Features and Limitations of Hive.

Features of Hive:

1. SQL-like Interface: Hive provides a familiar SQL-like query language called HiveQL, which allows users to interact with data using SQL-like syntax. This makes it easy for users who are familiar with SQL to work with Hive.

2. Scalability: Hive is designed to handle large-scale datasets. It leverages distributed processing frameworks like MapReduce and Tez, allowing it to process and analyze data in parallel across a cluster of machines.

3. Data Warehousing: Hive is well-suited for data warehousing tasks. It supports schema-on-read, allowing flexibility in data formats and structures. It also provides features like partitioning, bucketing, and indexing for efficient data organization and retrieval.

4. Hive Metastore: Hive incorporates a metastore that stores and manages metadata information about tables, partitions, columns, and other objects. This metadata helps in query optimization and provides a centralized catalog for managing data definitions.

5. Extensibility: Hive supports the integration of user-defined functions (UDFs) and user-defined aggregates (UDAs), allowing users to extend the functionality of Hive by writing custom code in various programming languages.

6. Data Integration: Hive integrates with various data storage systems and formats, including Hadoop Distributed File System (HDFS), Apache HBase, Amazon S3, and more. This enables data integration and analysis across different data sources.

7. Schema Evolution: Hive supports schema evolution, allowing changes to table schemas over time without requiring data migration or downtime. It enables the addition, modification, or deletion of columns in existing tables.

Limitations of Hive:

1. Batch Processing: Hive is primarily designed for batch processing and is not suitable for real-time or low-latency applications. The overhead of MapReduce or Tez job startup and the batch-oriented nature of these processing frameworks limit Hive's real-time processing capabilities.

2. High Latency: Hive queries can have high latency due to the distributed nature of data processing and the overhead of query planning and job execution. Interactive or ad-hoc queries may experience higher response times compared to traditional databases.

3. Lack of Full ACID Transactions: Hive does not natively support full ACID (Atomicity, Consistency, Isolation, Durability) transactions. While it provides transactional features like insert, update, and delete operations, it lacks the fine-grained control and isolation levels of traditional ACID databases.

4. Limited Indexing: Hive has limited support for indexing. While it provides basic indexing mechanisms like bitmap indexing and automatic indexing on partitioned columns, it lacks advanced indexing options like B-trees or bitmap indexes on non-partitioned columns.

5. Data Type Limitations: Hive has a limited set of built-in data types compared to traditional databases. It may not support some advanced data types or complex data structures, which can be a limitation for certain use cases.

6. Lack of Real-Time Data Ingestion: Hive is not designed for real-time data ingestion or streaming data processing. It is more suited for batch-oriented data processing and analysis.

7. Complexity: Hive's flexibility and power come with increased complexity. Setting up and managing a Hive environment requires expertise in configuring and optimizing the system, as well as understanding the underlying distributed processing frameworks.

1. How to create a Database in HIVE?

A database can be created in HIVE by using Create Database statement.

For example Create database Hive\_database;

1. How to create a table in HIVE?

A table in hive can be created by using Create table command.

Create table employee( emp\_id int ,

Name string)

Row format delimited

Fields terminated by ‘,’

Stored as textfile;

1. What do you mean by describe and describe extended and describe formatted with respect to database and table.

In the context of a database or table in Hive, the DESCRIBE statement is used to retrieve metadata information about the structure and properties of the database or table. There are three variations of the DESCRIBE statement: DESCRIBE, DESCRIBE EXTENDED, and DESCRIBE FORMATTED. Let's understand each one:

**DESCRIBE:**

Syntax: DESCRIBE database\_name; or DESCRIBE table\_name;

Usage: It provides a brief summary of the columns in a table or databases in a database. It displays the column names and their data types.

Example:

DESCRIBE mytable; - Provides a summary of the columns in the "mytable" table.

**DESCRIBE EXTENDED:**

Syntax: DESCRIBE EXTENDED database\_name; or DESCRIBE EXTENDED table\_name;

Usage: It provides more detailed information about the table or database, including column names, data types, and additional metadata like column comments and table location.

Example:

DESCRIBE EXTENDED mytable; - Provides detailed information about the "mytable" table, including column names, data types, comments, and other metadata.

**DESCRIBE FORMATTED:**

Syntax: DESCRIBE FORMATTED database\_name; or DESCRIBE FORMATTED table\_name;

Usage: It provides a detailed and formatted view of the table or database metadata, including information about the columns, partitioning, storage format, file location, and more.

Example:

**DESCRIBE FORMATTED** mytable; - Provides a formatted view of the metadata for the "mytable" table, including column details, partitioning information, storage format, and file location.

1. How to skip header rows from a table in Hive?

We can use skip.header.line.count =’1’ to skip header for a table.

1. What is a hive operator? What are the different types of hive operators?

In Hive, an operator is a symbol or keyword used to perform operations on data within a query. Operators allow you to manipulate and transform the data stored in Hive tables. They are an essential component of Hive's query language, HiveQL.

Hive operators can be categorized into different types based on the operations they perform. Here are the commonly used types of Hive operators:

**1. Relational Operators:**

- These operators are used to perform relational operations on tables or datasets. Examples include:

- `SELECT`: Retrieves specific columns or expressions from the table.

- `WHERE`: Filters rows based on specified conditions.

- `JOIN`: Combines rows from multiple tables based on specified join conditions.

- `GROUP BY`: Groups rows based on specified columns for aggregation.

- `ORDER BY`: Sorts the result set based on specified columns.

- `LIMIT`: Restricts the number of rows in the result set.

**2. Arithmetic Operators:**

- These operators are used to perform arithmetic operations on numeric data. Examples include:

- `+` (addition)

- `-` (subtraction)

- `\*` (multiplication)

- `/` (division)

- `%` (modulus)

- `=` (equality comparison)

- `<>` or `!=` (inequality comparison)

- `<` (less than)

- `>` (greater than)

- `<=` (less than or equal to)

- `>=` (greater than or equal to)

**3. Logical Operators:**

- These operators are used to perform logical operations on boolean values. Examples include:

- `AND` (logical AND)

- `OR` (logical OR)

- `NOT` (logical NOT)

**4. Unary Operators:**

- These operators operate on a single input. Examples include:

- `-` (unary minus)

- `NOT` (unary logical NOT)

These are just a few examples of Hive operators. There are other operators as well, such as string operators, aggregate functions, type conversion functions, etc., that can be used in HiveQL queries to manipulate and process data.

13.Explain about the Hive Built-In Functions

Hive provides a rich set of built-in functions that can be used in HiveQL queries to perform various operations on data. These functions can be categorized into different types based on the operations they perform. Here are some commonly used categories of Hive built-in functions:

1. Mathematical Functions:

- Examples: `abs()`, `ceil()`, `floor()`, `round()`, `sqrt()`, `power()`, `log()`, `exp()`, `sin()`, `cos()`, `tan()`, etc. These functions allow you to perform mathematical calculations on numeric data.

2. String Functions:

- Examples: `length()`, `concat()`, `substring()`, `trim()`, `lower()`, `upper()`, `regexp\_replace()`, `split()`, `translate()`, `substr()`, `instr()`, `lpad()`, `rpad()`, etc. These functions are used for manipulating and processing strings.

3. Date and Time Functions:

- Examples: `year()`, `month()`, `day()`, `hour()`, `minute()`, `second()`, `date\_format()`, `unix\_timestamp()`, `from\_unixtime()`, `current\_date()`, `current\_timestamp()`, `datediff()`, `date\_add()`, `date\_sub()`, etc. These functions are used for handling date and time values.

4. Conditional Functions:

- Examples: `if()`, `case`, `when`, `else`, `coalesce()`, `nullif()`, `nvl()`, `decode()`, etc. These functions allow you to perform conditional operations and handle null values.

5. Aggregate Functions:

- Examples: `sum()`, `avg()`, `min()`, `max()`, `count()`, `distinct()`, `collect\_list()`, `collect\_set()`, `percentile()`, `stddev()`, `variance()`, etc. These functions are used for aggregating data and calculating summary statistics.

6. Type Conversion Functions:

- Examples: `cast()`, `int()`, `double()`, `string()`, `boolean()`, `array()`, `struct()`, `map()`, etc. These functions allow you to convert data between different types.

7. Collection Functions:

- Examples: `size()`, `array\_contains()`, `map\_keys()`, `map\_values()`, `explode()`, `posexplode()`, `split()`, etc. These functions are used for working with collections such as arrays and maps.

1. Write hive DDL and DML commands.

DDL

1)CREATE TABLE table\_name (

column1 data\_type,

column2 data\_type,

...

)

[COMMENT 'description']

[PARTITIONED BY (partition\_column data\_type, ...)]

[STORED AS file\_format]

[LOCATION 'hdfs\_path'];

2.)ALTER TABLE table\_name

ADD COLUMNS (column1 data\_type, column2 data\_type, ...);

DML:

INSERT INTO table\_name [PARTITION (partition\_column=value, ...)]

VALUES (value1, value2, ...);

UPDATE table\_name

SET column1 = value1, column2 = value2

WHERE condition;

DELETE FROM table\_name

WHERE condition;

1. Explain about SORT BY, ORDER BY, DISTRIBUTE BY and CLUSTER BY in Hive.
2. Difference between "Internal Table" and "External Table" and Mention when to choose “Internal Table” and “External Table” in Hive?
3. Where does the data of a Hive table get stored?
4. Is it possible to change the default location of a managed table? 19.What is a metastore in Hive? What is the default database provided by

Apache Hive for metastore?

1. Why does Hive not store metadata information in HDFS?
2. What is a partition in Hive? And Why do we perform partitioning in Hive?
3. What is the difference between dynamic partitioning and static partitioning?
4. How do you check if a particular partition exists? 24.How can you stop a partition form being queried?

25.Why do we need buckets? How Hive distributes the rows into buckets? 26.In Hive, how can you enable buckets?

27.How does bucketing help in the faster execution of queries? 28.How to optimise Hive Performance? Explain in very detail.

1. What is the use of Hcatalog?
2. Explain about the different types of join in Hive.
3. Is it possible to create a Cartesian join between 2 tables, using Hive? 32.Explain the SMB Join in Hive?
4. What is the difference between order by and sort by which one we should use?
5. What is the usefulness of the DISTRIBUTED BY clause in Hive? 35.How does data transfer happen from HDFS to Hive?
6. Wherever (Different Directory) I run the hive query, it creates a new metastore\_db, please explain the reason for it?
7. What will happen in case you have not issued the command: ‘SET hive.enforce.bucketing=true;’ before bucketing a table in Hive?
8. Can a table be renamed in Hive?
9. Write a query to insert a new column(new\_col INT) into a hive table at a position before an existing column (x\_col)
10. What is serde operation in HIVE?
11. Explain how Hive Deserializes and serialises the data? 42.Write the name of the built-in serde in hive.
12. What is the need of custom Serde?
13. Can you write the name of a complex data type(collection data types) in Hive?
14. Can hive queries be executed from script files? How?
15. What are the default record and field delimiter used for hive text files? 47.How do you list all databases in Hive whose name starts with s?
16. What is the difference between LIKE and RLIKE operators in Hive?
17. How to change the column data type in Hive?
18. How will you convert the string ’51.2’ to a float value in the particular column?
19. What will be the result when you cast ‘abc’ (string) as INT? 52.What does the following query do?
    1. INSERT OVERWRITE TABLE employees
    2. PARTITION (country, state)
    3. SELECT ..., se.cnty, se.st
    4. FROM staged\_employees se;
20. Write a query where you can overwrite data in a new table from the existing table.
21. What is the maximum size of a string data type supported by Hive?

Explain how Hive supports binary formats.

1. What File Formats and Applications Does Hive Support?
2. How do ORC format tables help Hive to enhance its performance? 57.How can Hive avoid mapreduce while processing the query?
3. What is view and indexing in hive?
4. Can the name of a view be the same as the name of a hive table? 60.What types of costs are associated in creating indexes on hive tables? 61.Give the command to see the indexes on a table.

62. Explain the process to access subdirectories recursively in Hive queries. 63.If you run a select \* query in Hive, why doesn't it run MapReduce? 64.What are the uses of Hive Explode?

1. What is the available mechanism for connecting applications when we run Hive as a server?
2. Can the default location of a managed table be changed in Hive? 67.What is the Hive ObjectInspector function?
3. What is UDF in Hive?
4. Write a query to extract data from hdfs to hive.
5. What is TextInputFormat and SequenceFileInputFormat in hive. 71.How can you prevent a large job from running for a long time in a hive? 72.When do we use explode in Hive?
6. Can Hive process any type of data formats? Why? Explain in very detail
7. Whenever we run a Hive query, a new metastore\_db is created. Why? 75.Can we change the data type of a column in a hive table? Write a

complete query.

1. While loading data into a hive table using the LOAD DATA clause, how do you specify it is a hdfs file and not a local file ?
2. What is the precedence order in Hive configuration? 78.Which interface is used for accessing the Hive metastore? 79.Is it possible to compress json in the Hive external table ? 80.What is the difference between local and remote metastores? 81.What is the purpose of archiving tables in Hive?
3. What is DBPROPERTY in Hive?
4. Differentiate between local mode and MapReduce mode in Hive.