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

Hive is an open-source data warehousing and SQL-like query language developed for querying and managing large datasets stored in distributed storage systems such as Hadoop. It provides a familiar SQL-like interface to interact with the data and enables users to write queries using a language called Hive Query Language (HQL). Hive translates these HQL queries into MapReduce, Tez, or Spark jobs, which are then executed on a cluster of machines to process the data.

Hive is designed to facilitate data analysis and processing on large datasets that are typically stored in distributed file systems like Hadoop Distributed File System (HDFS) or Amazon S3. It provides tools and techniques for data summarization, querying, and analysis, making it easier for data analysts and data scientists to work with big data. The latest and present version of hive is release 4.0. 0-alpha-2.

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

Hive, which is a data warehousing infrastructure built on top of Apache Hadoop, is primarily designed for batch processing and analytical workloads rather than Online Transaction Processing (OLTP) systems. OLTP systems typically involve high-speed, real-time transactions with low-latency requirements, while Hive is optimized for large-scale data processing, batch querying, and analytics.

Here are a few reasons why Hive may not be suitable for OLTP systems:

1. Latency: Hive's architecture is based on a two-step process involving data ingestion and subsequent analysis. This introduces inherent latency in processing data, which is not suitable for OLTP systems that require near-real-time or real-time responses.

2. Data Model: Hive is built on a schema-on-read approach, where the data schema is applied at query time rather than during data ingestion. This design makes it flexible for handling unstructured and semi-structured data, but it also adds overhead for handling individual record-level updates, inserts, or deletes typically required in OLTP systems.

3. Indexing: Hive relies on indexing techniques such as Bloom filters and bitmap indexes, which are suitable for efficient batch processing and analytical queries. However, OLTP systems often require fast access to specific records and rely on primary and secondary indexes for efficient data retrieval and modification. Hive's indexing mechanisms are not optimized for OLTP workloads.

4. ACID Compliance: ACID (Atomicity, Consistency, Isolation, Durability) properties are crucial for maintaining data integrity in OLTP systems. While Hive introduced support for ACID transactions in recent versions, it is still not as robust and optimized as traditional OLTP databases like MySQL or PostgreSQL.

5. Concurrency: OLTP systems typically have high levels of concurrent read and write operations. Hive's design favors batch processing and is less efficient in handling concurrent small-scale transactions that are characteristic of OLTP workloads.

If you require an OLTP system, it is advisable to consider traditional relational databases or other specialized systems designed explicitly for handling real-time transactional workloads.

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

Hive and RDBMS (Relational Database Management Systems) are different in several ways. Here are some key differences between them:

1. Data Model: Hive is designed to work with structured and semi-structured data stored in distributed file systems like Hadoop Distributed File System (HDFS). It follows a schema-on-read approach, meaning the structure is applied when the data is queried. On the other hand, RDBMS follows a schema-on-write approach, where the data must conform to a predefined schema before it is stored in the database.

2. Query Language: Hive uses a SQL-like language called HiveQL (Hive Query Language) to interact with data. It is similar to SQL but has some differences to accommodate the distributed nature of the underlying data. RDBMS, on the other hand, uses standard SQL for querying and manipulating data.

3. Scalability: Hive is designed for scalability and can handle large amounts of data by leveraging distributed computing frameworks like Hadoop. It can process data in parallel across multiple nodes. RDBMS, while it can handle significant amounts of data, may face challenges in scaling to the same extent as Hive due to its centralized architecture.

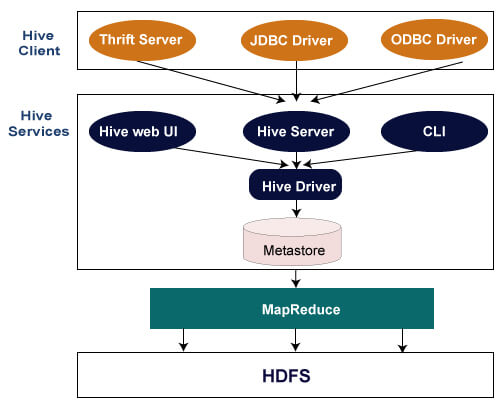
4. Performance: RDBMS is optimized for transactional processing and is generally faster for transactional workloads involving real-time queries or frequent updates. Hive, on the other hand, is optimized for batch processing and is suitable for analytics and data processing tasks involving large volumes of data.

Regarding ACID (Atomicity, Consistency, Isolation, Durability) transactions, Hive does not natively support them. ACID transactions guarantee that a series of database operations are performed atomically (all or nothing), consistently, in isolation from other transactions, and durably stored. The reason Hive does not support ACID transactions is primarily due to its design priorities and trade-offs for scalability and performance.

Hive is built on top of Hadoop, which is a distributed file system, and it prioritizes scalability and fault-tolerance over strict transactional guarantees. ACID transactions require coordination and synchronization across distributed nodes, which can introduce significant overhead and potentially impact performance and scalability. Hive is more focused on large-scale batch processing and data analysis, where strict transactional guarantees are not always necessary.

However, it's worth mentioning that there are extensions and tools available for Hive, such as Apache HBase, that provide support for ACID transactions. These extensions integrate Hive with other systems that can provide the necessary transactional capabilities if required for specific use cases.

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



Hive is a data warehousing infrastructure built on top of Hadoop, designed to provide scalable and high-performance querying and analysis of large datasets. It provides a high-level interface known as HiveQL, which is similar to SQL, allowing users to write queries and perform data analysis using a familiar syntax.

The architecture of Hive consists of several components that work together to enable efficient data processing and analysis. Here are the key components:

1. Hive Clients: Hive clients are the applications or interfaces used to interact with the Hive system. They can be command-line tools, graphical user interfaces (GUI), or programming language APIs such as JDBC or ODBC. These clients submit queries and commands to Hive for processing.

2. Metastore: The Metastore is a central component of Hive that stores metadata about the data stored in Hive. It contains information about tables, partitions, columns, data types, and other structural details. The Metastore can be backed by a traditional relational database management system (RDBMS) such as MySQL, or it can use a lightweight embedded database like Derby.

3. Hive Query Language (HiveQL): HiveQL is a SQL-like language that allows users to write queries and perform data manipulation operations on the data stored in Hive. It provides a familiar syntax to SQL users and supports various SQL operations like SELECT, INSERT, UPDATE, DELETE, JOIN, and GROUP BY.

4. Hive Compiler: The Hive compiler is responsible for parsing and analyzing the queries submitted by users. It performs semantic analysis, query optimization, and query plan generation. The compiler translates HiveQL queries into a directed acyclic graph (DAG) representation called the Query Plan.

5. Query Execution Engine: The Query Execution Engine executes the query plans generated by the Hive compiler. Hive supports different execution engines, such as MapReduce, Tez, or Spark, which are responsible for executing the tasks required to process the data.

6. Hadoop Distributed File System (HDFS): Hive leverages HDFS as its underlying storage system. HDFS provides a distributed and scalable file system that can store large amounts of data across multiple machines. Hive uses HDFS to store the data in a distributed manner, enabling high data availability and fault tolerance.

7. Hive Warehouse: The Hive Warehouse is the physical storage location where the data is stored in Hive. It is typically an HDFS directory or a set of directories where the data files are stored in a structured or semi-structured format, such as CSV, Parquet, or ORC.

8. Hive SerDe: SerDe stands for Serializer/Deserializer. Hive uses SerDes to read and write data from various file formats stored in the Hive Warehouse. SerDes handle the serialization and deserialization of data between the file format and the internal representation used by Hive.

9. Hive UDFs: User-Defined Functions (UDFs) allow users to define their own custom functions in Hive. UDFs can be written in programming languages like Java or Python and enable users to extend the functionality of Hive by incorporating their own logic into queries.

These components work together to provide a powerful and scalable data warehousing solution with Hive, allowing users to perform complex analytics and processing tasks on large datasets stored in Hadoop.

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 executing queries in Apache Hive, a data warehouse infrastructure built on top of Hadoop. It translates HiveQL (a SQL-like query language) into a series of MapReduce, Tez, or Spark jobs, which are then executed on a Hadoop cluster.

The components of a Hive query processor include:

1. Parser: The parser parses the HiveQL statements and generates an abstract syntax tree (AST). It performs syntactic and semantic analysis to ensure the correctness of the statements.

2. Semantic Analyzer: The semantic analyzer processes the AST generated by the parser and performs semantic analysis. It checks for table and column existence, resolves column names, and performs type checking.

3. Query Planner: The query planner takes the validated AST and generates an execution plan. It determines the optimal execution strategy based on the query and the underlying data. The planner decides whether to use MapReduce, Tez, or Spark for query execution.

4. Compiler: The compiler takes the execution plan generated by the query planner and compiles it into a series of MapReduce, Tez, or Spark jobs. It optimizes the execution plan to minimize resource usage and improve performance.

5. Execution Engine: The execution engine is responsible for executing the compiled jobs on the Hadoop cluster. It manages the scheduling and coordination of the tasks and monitors their progress.

6. Metastore: The metastore is a central repository that stores the metadata information about tables, partitions, and schemas. It provides the query processor with access to the metadata required for query execution, such as table locations and column statistics.

7. Input/Output Formats: Hive supports various input and output formats for reading and writing data. The query processor interacts with these formats to read input data from Hadoop Distributed File System (HDFS) or other storage systems and write query results to the desired output formats.

Overall, the Hive query processor plays a crucial role in transforming HiveQL queries into executable jobs and managing the execution on a Hadoop cluster, allowing users to analyze and process large datasets using SQL-like queries.

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

Hive, a data warehousing and SQL-like query language for Apache Hadoop, can be operated in three different modes:

1. Local Mode: In this mode, Hive runs entirely on the local machine without any connection to a Hadoop cluster. It uses the local file system for storage and processing. Local mode is primarily used for development, testing, and debugging purposes when dealing with small datasets.

2. MapReduce Mode: This is the default mode of operation for Hive. In MapReduce mode, Hive interacts with a Hadoop cluster, leveraging the MapReduce framework for distributed processing. Hive queries are translated into MapReduce jobs, which are then executed on the cluster. This mode allows Hive to handle large-scale datasets and provides scalability and fault tolerance.

3. Spark Mode: Hive can also be operated in Spark mode, leveraging Apache Spark for distributed data processing. Spark mode provides faster query execution compared to MapReduce mode due to Spark's in-memory processing capabilities. It allows users to take advantage of Spark's rich set of libraries and supports real-time streaming, machine learning, and graph processing. Spark mode is particularly beneficial for interactive and iterative data analysis workloads.

It's worth mentioning that Hive can also integrate with other execution engines like Tez and LLAP (Low Latency Analytical Processing) for optimized query performance.

1. **Features and Limitations of Hive.**

1. Open-source: Apache Hive is an open-source tool. We can use it free of cost.

2. Query large datasets: Hive can query and manage huge datasets stored in Hadoop Distributed

File System.

3. Multiple-users: Multiple users can query the data using Hive Query Language simultaneously.

4. Backward compatible: Apache Hive perfectly fits the low level interface requirement of Apache Hadoop.

5. Partitioning and Bucketing: Apache Hive supports partitioning and bucketing of data at the table level to improve performance.

6. File-formats: Hive provides support for various file formats such as textFile, ORC, Avro Files, SequenceFile, Parquet, RCFile, LZO Compression etc.

7. Hive Query Language: Hive uses Hive Query Language which is similar to SQL. We do not require any knowledge of programming languages to work with Hive. Only the knowledge of basic SQL query is enough to work with Hive.

8. Built-In function: Hive provides various Built-In functions.

9. User-Defined Functions: It also provides support for User-Defined Functions for the tasks like data cleansing and filtering. We can define UDFs according to our requirements.

10. External Table: Apache Hive supports external tables. This allows us to process data without actually storing data in HDFS.

11. Fast: Hive is a fast, scalable, extensible tool and uses familiar concepts.

12. Warehouse: Apache Hive is a distributed data warehouse tool.

13. Table Structure: Table structure in Hive is similar to table structure in RDBMS.

14. ETL support: Hive supports ETL operations. Hive is an effective ETL tool.

15. Storage: Hive allows us to access files stored in HDFS and other similar data storage systems such as HBase.

16. OLAP: Hive is designed for OLAP (Online Analytical Processing).

17. Client application: Hive can support client applications written in PHP, Python, Java, C++ and Ruby.

18. Rule Based Optimizer: Hive has a rule-based optimizer for optimizing logical plans.

19. Ad-hoc queries: Hive allows us to run Ad-hoc queries which are the loosely typed command or query whose value depends on some variable for the data analysis.

20. Data Visualization: Hive can be used for Data Visualization. Integrating Hive with Apache Tez will provide the real time processing capabilities.

***Some of the limitations of Apache Hive are:***

1. Hive is not designed for the OLTP (Online transaction processing). We can use it for OLAP.

2. It does not offer real-time queries.

3. It provides limited subquery support.

4. Latency of Hive is generally very high.

1. **How to create a Database in HIVE?**

To create a database in Hive, you can follow these steps:

1. Start by opening a Hive session or connecting to the Hive command line interface (CLI).

2. Once you are in the Hive CLI, execute the following command to create a database:

```

**CREATE DATABASE database\_name;**

```

Replace `database\_name` with the desired name for your database.

3. By default, Hive creates the database in the Hive warehouse directory specified in the Hive configuration. However, you can specify a custom location for the database files using the following command:

```

**CREATE DATABASE database\_name LOCATION 'hdfs://your\_hdfs\_path';**

```

Replace `your\_hdfs\_path` with the HDFS path where you want to store the database files. If you don't specify a location, Hive will use the default warehouse directory.

4. Once you execute the command, Hive will create the database and confirm the operation with a message.

5. You can verify the existence of the newly created database by executing the following command:

```

**SHOW DATABASES;**

```

This will display a list of all the databases in Hive, including the one you just created.

1. **How to create a table in HIVE?**

To create a table in Hive, you can use the HiveQL language, which is similar to SQL. Here's an example of how to create a table in Hive:

**```**

**CREATE TABLE my\_table (**

**column1 INT,**

**column2 STRING,**

**column3 DOUBLE**

**);**

**```**

***Let's break down the above example:***

**- `CREATE TABLE`** is the statement used to create a new table in Hive.

**- `my\_table`** is the name of the table you want to create. You can choose any name you like.

**- `(column1 INT, column2 STRING, column3 DOUBLE)`** defines the columns in the table. In this example, we have three columns: **`column1`** of type **`INT`,** **`column2`** of type **`STRING`,** and **`column3`** of type **`DOUBLE`.**

You can customize the table schema according to your requirements by adding or removing columns and specifying the appropriate data types.

Once you've executed the **`CREATE TABLE`** statement, Hive will create an empty table with the specified schema. You can then insert data into the table using **`INSERT INTO`** statements or load data from external sources.

It's important to note that Hive is a data warehousing system built on top of Apache Hadoop. Therefore, it's designed for large-scale data processing and works best with batch processing jobs.

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

In the context of databases and tables, the terms "describe," "describe extended," and "describe formatted" are typically used in SQL to retrieve metadata about database objects such as tables, views, or columns. Let me explain each term:

1. DESCRIBE: The "DESCRIBE" command provides a brief summary of the structure of a particular database object, usually a table. When you execute the "DESCRIBE" command followed by the name of a table, it returns information about the columns or fields within that table, including their names, data types, and any constraints or indexes associated with them. This command is also commonly represented by "DESC."

**Example:**

**```**

**DESCRIBE table\_name;**

**```**

**Output:**

**```**

**+-------------+-------------+------+-----+---------+-------+**

**| Field | Type | Null | Key | Default | Extra |**

**+-------------+-------------+------+-----+---------+-------+**

**| column1 | datatype | YES | | NULL | |**

**| column2 | datatype | NO | | | |**

**| column3 | datatype | YES | | NULL | |**

**+-------------+-------------+------+-----+---------+-------+**

**```**

2. DESCRIBE EXTENDED: The "DESCRIBE EXTENDED" command provides more detailed information about a database object compared to the standard "DESCRIBE" command. It includes additional details such as column statistics, location of data files, and other related properties. The extended description is useful when you need a more comprehensive understanding of the object's structure and attributes.

**Example:**

**```**

**DESCRIBE EXTENDED table\_name;**

**```**

**Output:**

**```**

**+-------------+-------------+------+-----+---------+-------+-----------------------------+**

**| Field | Type | Null | Key | Default | Extra | Comment |**

**+-------------+-------------+------+-----+---------+-------+-----------------------------+**

**| column1 | datatype | YES | | NULL | | |**

**| column2 | datatype | NO | | | | |**

**| column3 | datatype | YES | | NULL | | |**

**+-------------+-------------+------+-----+---------+-------+-----------------------------+**

**| Detailed information about column statistics, data files, and other properties |**

**+-----------------------------------------------------------------------------------+**

**```**

3. DESCRIBE FORMATTED: The "DESCRIBE FORMATTED" command provides a more formatted and readable output of the metadata information for a database object. It presents the information in a structured format, making it easier to analyze and understand. The formatted description is often helpful when you want a well-organized presentation of the object's properties and attributes.

**Example:**

**```**

**DESCRIBE FORMATTED table\_name;**

**```**

**Output:**

**```**

**+-------------+-------------+------+-----+---------+-------+**

**| Field | Type | Null | Key | Default | Extra |**

**+-------------+-------------+------+-----+---------+-------+**

**| column1 | datatype | YES | | NULL | |**

**| column2 | datatype | NO | | | |**

**| column3 | datatype | YES | | NULL | |**

**+-------------+-------------+------+-----+---------+-------+**

**```**

Please note that the actual output of these commands might vary depending on the database management system (DBMS) you are using, as different DBMS may have their own variations or alternative commands to achieve similar results.

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

In Hive, you can skip header rows from a table by using the **`tblproperties`** clause with the `**skip.header.line.count**` parameter. This parameter specifies the number of header lines to skip when loading data into a table. Here's an example of how to skip header rows from a table in Hive:

**```sql**

**CREATE TABLE my\_table (**

**column1 STRING,**

**column2 INT,**

**column3 FLOAT**

**)**

**ROW FORMAT DELIMITED**

**FIELDS TERMINATED BY ','**

**STORED AS TEXTFILE**

**TBLPROPERTIES (**

**"skip.header.line.count"="1"**

**);**

**```**

In this example, we assume that the header row contains one line. By setting **`skip.header.line.count`** to 1, Hive will ignore the first line of the input file, assuming it is a header row.

You can adjust the value of **`skip.header.line.count`** based on the number of header rows you want to skip. If there are multiple header rows, set the value accordingly.

After creating the table with the **`skip.header.line.count`** property, you can load data into it using the **`LOAD DATA INPATH`** or **`INSERT INTO TABLE`** statement, and Hive will automatically skip the specified number of header lines.

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

Apache Hive provides various Built-in operators for data operations to be implemented on the tables present inside Apache Hive warehouse. Hive operators are used for mathematical operations on operands. It returns specific value as per the logic applied.

There are different types of hive operators: -

1. Relational Operators
2. Arithmetic Operators
3. Logical Operators
4. Complex Operators
5. String Operators
6. **Explain about the hive built-in functions**

The Hive Built-in functions are categorized as Mathematical function, Collection function, Type conversion function, Date function, Conditional function, and String function. Hive is a data warehouse infrastructure built on top of Hadoop that provides a query language called HiveQL, which is similar to SQL. Hive also includes a wide range of built-in functions that can be used in queries to manipulate and transform data. These built-in functions can be categorized into various types, including string functions, mathematical functions, date and time functions, conditional functions, and aggregate functions.

Here are some commonly used categories of Hive built-in functions:

1. String Functions: Hive provides a rich set of functions to work with strings, such as concatenating strings (`concat`), extracting substrings (`substr`), converting case (`lower`, `upper`), trimming spaces (`trim`), replacing substrings (`regexp\_replace`), and many more.

2. Mathematical Functions: Hive offers various mathematical functions for performing calculations on numeric data, including basic arithmetic functions (`+, -, \*, /`), rounding functions (`round`, `ceil`, `floor`), trigonometric functions (`sin`, `cos`, `tan`), logarithmic functions (`log`, `log10`), and more.

3. Date and Time Functions: Hive supports a range of functions to handle date and time data, such as extracting components from dates (`year`, `month`, `day`, `hour`, `minute`, `second`), formatting dates (`date\_format`), calculating differences between dates (`datediff`, `months\_between`), and more.

4. Conditional Functions: These functions allow you to perform conditional checks and return values based on specified conditions. Examples include the `if`, `case`, and `when` functions, which enable you to implement if-else logic and perform conditional computations within your queries.

5. Aggregate Functions: Hive provides a set of aggregate functions to summarize data across groups or entire datasets. These include functions like `sum`, `avg`, `min`, `max`, `count`, `group\_concat`, and more. Aggregate functions are commonly used with the `GROUP BY` clause to generate summary statistics or perform data aggregations.

These are just a few examples of the built-in functions available in Hive. The complete list of functions can be found in the Hive documentation. Additionally, Hive also supports user-defined functions (UDFs) and user-defined aggregate functions (UDAFs), which allow you to extend the functionality of Hive by writing your own custom functions in Java, Python, or other languages.

1. **Write hive DDL and DML commands**

Below are examples of Hive Data Definition Language (DDL) and Data Manipulation Language (DML) commands:

***DDL Commands:***

1. Create a database:

```

**CREATE DATABASE database\_name;**

```

2. Use a database:

```

**USE database\_name;**

```

3. Create an external table:

```

**CREATE EXTERNAL TABLE table\_name (**

**column1 data\_type,**

**column2 data\_type,**

...

**)**

**LOCATION 'hdfs\_path';**

```

4. Create a managed table:

```

**CREATE TABLE table\_name (**

**column1 data\_type,**

**column2 data\_type,**

**...**

**);**

**```**

5. Alter a table to add/drop columns:

**```**

**ALTER TABLE table\_name**

**ADD COLUMN new\_column data\_type;**

**ALTER TABLE table\_name**

**DROP COLUMN column\_name;**

**```**

6. Rename a table:

**```**

**ALTER TABLE table\_name**

**RENAME TO new\_table\_name;**

**```**

7. Drop a table:

**```**

**DROP TABLE table\_name;**

**```**

***DML Commands:***

1. Load data into a table:

**```**

**LOAD DATA [LOCAL] INPATH 'input\_path' [OVERWRITE] INTO TABLE table\_name;**

**```**

2. Insert data into a table:

**```**

**INSERT INTO TABLE table\_name**

**VALUES (value1, value2, ...);**

3. Insert data into a partitioned table:

**```**

**INSERT INTO TABLE table\_name PARTITION (partition\_column)**

**VALUES (value1, value2, ..., partition\_value);**

**```**

4. Update data in a table:

**```**

**UPDATE table\_name**

**SET column\_name = new\_value**

**WHERE condition;**

**```**

5. Delete data from a table:

**```**

**DELETE FROM table\_name**

**WHERE condition;**

**```**

6. Create a new table based on a query result:

**```**

**CREATE TABLE new\_table\_name AS**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

**```**

These are just a few examples of the DDL and DML commands in Hive. Hive supports many more commands and options for managing and manipulating data in Hadoop distributed file systems. ```

5. Delete data from a table:

**```**

**DELETE FROM table\_name**

**WHERE condition;**

**```**

6. Create a new table based on a query result:

**```**

**CREATE TABLE new\_table\_name AS**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

**```**

These are just a few examples of the DDL and DML commands in Hive. Hive supports many more commands and options for managing and manipulating data in Hadoop distributed file systems.

1. **Explain about SORT BY, ORDER BY, DISTRIBUTE BY and CLUSTER BY in Hive.**

These Sort By, Order By, Distribute By, and Cluster By clauses are available in the hive query language and we can use them to distribute and order the output data in different ways. The SORT BY and ORDER BY clauses are used to define the order of the output data. However, DISTRIBUTE BY and CLUSTER BY clauses are used to distribute the data to multiple reducers based on the key columns. We can use Sort by or Order by or Distribute by or Cluster by clauses in a hive SELECT query to get the output data in the desired order.

1. **Difference between "Internal Table" and "External Table" and Mention when to choose “Internal Table” and “External Table” in Hive?**

***Hive Internal Table :-***

Hive owns the data for the internal tables. It is the default table in Hive. When the user creates a table in Hive without specifying it as external, then by default, an internal table gets created in a specific location in HDFS. By default, an internal table will be created in a folder path similar to /user/hive/warehouse directory of HDFS. We can override the default location by the location property during table creation. If we drop the managed table or partition, the table data and the metadata associated with that table will be deleted from the HDFS.

***Hive External Table:-***

Hive does not manage the data of the External table. We create an external table for external use as when we want to use the data outside the Hive. External tables are stored outside the warehouse directory. They can access data stored in sources such as remote HDFS locations or Azure Storage Volumes. Whenever we drop the external table, then only the metadata associated with the table will get deleted, the table data remains untouched by Hive. We can create the external table by specifying the EXTERNAL keyword in the Hive create table statement.

1. **Where does the data of a Hive table get stored?**

Hive stores data at the HDFS location /user/hive/warehouse folder if not specified a folder using the LOCATION clause while creating a table. Hive is a data warehouse database for Hadoop, all database and table data files are stored at HDFS location /user/hive/warehouse by default, you can also store the Hive data warehouse files either in a custom location on HDFS, S3, or any other Hadoop compatible file systems.

1. **Is it possible to change the default location of a managed table?**

Yes, you can do it by using the clause – LOCATION '<hdfs\_path>' we can change the default location of a managed table.

1. **What is a metastore in Hive? What is the default database provided by Apache Hive for metastore?**

Hive Metastore is a component in Hive that stores the catalog of the system that contains the metadata about Hive create columns, Hive table creation, and partitions. Metadata is mostly stored in the traditional form of RDBMS. The Apache Hives make use of the Derby databases to store the metadata.

Derby embedded JDBC driver class. Unit test data goes in here on your local filesystem. If you want to run Derby as a network server so the metastore can be accessed from multiple nodes, see Hive Using Derby in Server Mode.

1. **Why does Hive not store metadata information in HDFS?**

Hive stores metadata information in the metastore using RDBMS instead of HDFS. The reason for choosing RDBMS is to achieve low latency as HDFS read/write operations are time consuming processes.

1. **What is a partition in Hive? And why do we perform partitioning in Hive?**

The partitioning in Hive means dividing the table into some parts based on the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

1. **What is the difference between dynamic partitioning and static partitioning?**

***Hive Static Partitioning :-***

* Insert input data files individually into a partition table is Static Partition.
* Usually when loading files (big files) into Hive tables static partitions are preferred.
* Static Partition saves your time in loading data compared to dynamic partition.
* You “statically” add a partition in the table and move the file into the partition of the table.
* We can alter the partition in the static partition.
* You can get the partition column value from the filename, day of date etc without reading the whole big file.
* If you want to use the Static partition in the hive you should set property set hive.mapred.mode = strict This property set by default in hive-site.xml
* Static partition is in Strict Mode.
* You should use where clause to use limit in the static partition.
* You can perform Static partition on Hive Manage table or external table.

***Hive Dynamic Partitioning :-***

* Single insert to partition table is known as a dynamic partition.
* Usually, dynamic partition loads the data from the non-partitioned table.
* Dynamic Partition takes more time in loading data compared to static partition.
* When you have large data stored in a table then the Dynamic partition is suitable.
* If you want to partition a number of columns but you don’t know how many columns then also dynamic partition is suitable.
* Dynamic partition there is no required where clause to use limit.
* we can’t perform alter on the Dynamic partition.
* You can perform dynamic partition on hive external table and managed table.
* If you want to use the Dynamic partition in the hive then the mode is in non-strict mode.
* Here are Hive dynamic partition properties you should allow.

1. **How do you check if a particular partition exists?**

SHOW PARTITIONS table\_name

PARTITION(partitioned\_column=’partition\_value’)

1. **How can you stop a partition form being queried?**

By using the ENABLE OFFLINE clause with ALTER TABLE statement.

1. **Why do we need buckets? How Hive distributes the rows into buckets?**

The bucketing in Hive is a data organizing technique. It is similar to partitioning in Hive with an added functionality that it divides large datasets into more manageable parts known as buckets. So, we can use bucketing in Hive when the implementation of partitioning becomes difficult. However, we can also divide partitions further in buckets.

* The concept of bucketing is based on the hashing technique.
* Here, modules of current column value and the number of required buckets is calculated (let say, F(x) % 3).
* Now, based on the resulted value, the data is stored into the corresponding bucket.

1. **In Hive, how can you enable buckets?**

The command set hive. enforce. bucketing = true; allows the correct number of reducers and the cluster by column to be automatically selected based on the table. Otherwise, you would need to set the number of reducers to be the same as the number of buckets as in set mapred.

1. **How does bucketing help in the faster execution of queries?**

With bucketing in Hive, you can decompose a table data set into smaller parts, making them easier to handle. Bucketing allows you to group similar data types and write them to one single file, which enhances your performance while joining tables or reading data. This is a big reason why we use bucketing with partitioning most of the time.

1. **How to optimise Hive Performance? Explain in very detail.**

Performance tuning is key to optimizing a Hive query. First, tweak your data through partitioning, bucketing, compression, etc. Improving the execution of a hive query is another Hive query optimization technique. You can do this by using Tez, avoiding skew, and increasing parallel execution. Lastly, sampling and unit testing can help optimize a query by allowing you to see (and solve) problems on a smaller scale.

However, to run queries on petabytes of data we all know that hive is a query language which is similar to SQL built on Hadoop ecosystem. So, there are several Hive optimization techniques to improve its performance which we can implement when we run our hive queries.

1. *Tez-Execution Engine in Hive*
2. *Usage of Suitable File Format in Hive*
3. *Hive Partitioning*
4. *Bucketing in Hive*
5. *Vectorization In Hive*
6. *Cost-Based Optimization in Hive (CBO)*
7. *Hive Indexing.*
8. **What is the use of HCatalog?**

HCatalog is a table storage management tool for Hadoop that exposes the tabular data of Hive metastore to other Hadoop applications. It enables users with different data processing tools (Pig, MapReduce) to easily write data onto a grid. HCatalog ensures that users don’t have to worry about where or in what format their data is stored.

HCatalog is a tool that allows you to access Hive metastore tables within Pig, Spark SQL, and/or custom MapReduce applications. HCatalog has a REST interface and command line client that allows you to create tables or do other operations. You then write your applications to access the tables using HCatalog libraries.

1. **Explain about the different types of join in Hive.**

In Hive, which is a data warehouse infrastructure built on top of Apache Hadoop, there are several types of joins that can be used to combine data from multiple tables. These join types include:

1. Inner Join:

An inner join returns only the rows that have matching values in both tables being joined. It discards the non-matching rows from both tables. The result set contains only the rows where the join condition is satisfied.

1. Left Join (or Left Outer Join):

A left join returns all the rows from the left table and the matching rows from the right table. If there is no match, NULL values are returned for the right table. In other words, it retains all the rows from the left table and includes matching rows from the right table.

1. Right Join (or Right Outer Join):

A right join is the opposite of a left join. It returns all the rows from the right table and the matching rows from the left table. If there is no match, NULL values are returned for the left table. It retains all the rows from the right table and includes matching rows from the left table.

1. Full Join (or Full Outer Join):

A full join returns all the rows from both tables, including both the matching and non-matching rows. If there is no match, NULL values are returned for the non-matching side. It combines the results of left and right joins, creating a union of the two result sets.

1. Left Semi Join:

A left semi join returns all the rows from the left table for which there is a match in the right table. It doesn't return any columns from the right table. It is useful for filtering rows in the left table based on a condition in the right table.

1. Left Anti Join:

A left anti join returns all the rows from the left table for which there is no match in the right table. It doesn't return any columns from the right table. It is useful for filtering rows in the left table based on the absence of a match in the right table.

These join types can be specified in Hive using the JOIN keyword along with the appropriate join type (e.g., INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN). The join condition is typically specified using the ON keyword followed by the join criteria.

1. **Is it possible to create a Cartesian join between 2 tables, using Hive?**

Cross join, also known as Cartesian product, is a way of joining multiple tables in which all the rows or tuples from one table are paired with the rows and tuples from another table. For example, if the left-hand side table has 10 rows and the right-hand side table has 13 rows then the result set after joining the two tables will be 130 rows. That means all the rows from the left-hand side table (having 10 rows) are paired with all the tables from the right-hand side table (having 13 rows).

If there is a WHERE clause in the SQL statement that includes a cross join, then first the cross join takes place and then the result set is filtered out with the help of the WHERE clause. This means cross joins are not an efficient and optimized way of joining the tables.

1. **Explain the SMB Join in Hive?**

In Hive, while each mapper reads a bucket from the first table and the corresponding bucket from the second table, in SMB join. Basically, then we perform a merge sort join feature. Moreover, we mainly use it when there is no limit on file or partition or table join.

Also, when the tables are large, we can use Hive Sort Merge Bucket join. However, using the join columns, all join the columns are bucketed and sorted in SMB. Although, make sure in SMB join all tables should have the same number of buckets.

There are several scenarios when we can use Hive Sort Merge Bucket Join:

* While all tables are Large.
* Also, while all tables are bucketed using the join columns.
* While by using the join columns, Sorted.
* Also, when the number of buckets is same as the number of all tables.

1. **What is the difference between order by and sort by which one we should use?**

In most database management systems (DBMS), the terms "ORDER BY" and "SORT BY" are used interchangeably to achieve similar functionality. However, there might be some subtle differences depending on the context and the specific DBMS being used.

Generally speaking, both "ORDER BY" and "SORT BY" are used to arrange the result set of a query in a specified order. They allow you to sort the data based on one or more columns. The columns can be sorted in ascending (default) or descending order.

The choice between "ORDER BY" and "SORT BY" depends on the DBMS you are using and the specific syntax it supports. Here are a few examples:

1. SQL (Structured Query Language):

- "ORDER BY" is the standard syntax used in most SQL implementations, including popular ones like MySQL, PostgreSQL, and Oracle.

- Example: `SELECT \* FROM table\_name ORDER BY column\_name;`

2. Apache Spark:

- "SORT BY" is the preferred syntax in Spark's DataFrame API and SQL syntax.

- Example: `SELECT \* FROM table\_name SORT BY column\_name;`

It's important to consult the documentation of the specific DBMS or framework you are using to determine the correct syntax.

In summary, both "ORDER BY" and "SORT BY" are commonly used to sort query results, but the choice between them depends on the specific DBMS or framework you are working with.

1. **How does data transfer happen from HDFS to Hive?**

If data is already present in HDFS then the user need not LOAD DATA that moves the files to the /user/hive/warehouse/. So, the user just has to define the table using the keyword external that creates the table definition in the hive metastore.

Create external table table\_name (

id int,

myfields string

)

location '/my/location/in/hdfs';

1. **Wherever (Different Directory) I run the hive query, it creates a new metastore\_db, please explain the reason for it?**

When running Hive in embedded mode, it creates a local metastore. When you run the query, it first checks whether a metastore already exists or not. The property javax.jdo.option.ConnectionURL defined in the hive-site.xml has a default value jdbc: derby: databaseName=metastore\_db; create=true.

The value implies that embedded derby will be used as the Hive metastore and the location of the metastore is metastore\_db which will be created only if it does not exist already. The location metastore\_db is a relative location so when you run queries from different directories it gets created at all places from wherever you launch hive. This property can be altered in the hive-site.xml file to an absolute path so that it can be used from that particular location instead of creating multiple metastore\_db subdirectory multiple times.

1. **What will happen in case you have not issued the command: ‘SET hive.enforce.bucketing=true;’ before bucketing a table in Hive?**

The command: *‘SET hive.enforce.bucketing=true;’* allows you to have the correct number of reducer while using ‘CLUSTER BY’ clause for bucketing a column. In case it’s not done, one may find the number of files generated in the table directory to be unequal to the number of buckets. As an alternative solution, one may also set the number of reducer equal to the number of buckets by using *set mapred.reduce.task = num\_bucket*.

1. **Can a table be renamed in Hive?**

Yes, you can change a table name in Hive. You can rename a table name by using: Alter Table table\_name RENAME TO new\_name.

1. **Write a query to insert a new column(new\_col INT) into a hive table at a position before an existing column (x\_col)**

ALTER TABLE table\_name

CHANGE COLUMN new\_col INT

BEFORE x\_col

1. **What is serde operation in HIVE?**

A SerDe is a short name for a Serializer Deserializer. Hive uses SerDe to read and write data from tables. An important concept behind Hive is that it DOES NOT own the Hadoop File System format that data is stored in. Users are able to write files to HDFS with whatever tools/mechanism takes their fancy("CREATE EXTERNAL TABLE" or "LOAD DATA INPATH," ) and use Hive to correctly "parse" that file format in a way that can be used by Hive. A SerDe is a powerful (and customizable) mechanism that Hive uses to "parse" data stored in HDFS to be used by Hive.

SerDe - Serializer, Deserializer instructs hive on how to process a record (Row). Hive enables semi-structured (XML, Email, etc) or unstructured records (Audio, Video, etc) to be processed also. For Example If you have 1000 GB worth of RSS Feeds (RSS XMLs). You can ingest those to a location in HDFS. You would need to write a custom SerDe based on your XML structure so that Hive knows how to load XML files to Hive tables or other way around.

So how does Hive understand these different kinds of data format? The answer is SerDe. To understand how things work, lets break down things into the following sections:

Serialization and Deserialization

Hive Row Format

Map-reduce Input/Output Format

***Serialization and Deserialization***

Before diving deep into specifics of any Hive or Map-reduce its important to understand the above terms. The above two terms have been picked up from Java wherein

Serialization — Process of converting an object in memory into bytes that can be stored in a file or transmitted over a network.

Deserialization — Process of converting the bytes back into an object in memory.

Java understands objects and hence object is a deserialized state of data. When you use the same concept, Hive understands “columns” and hence if given a “row” of data, the task of converting that data into columns is the Deserialization part of Hive SerDe. In short

“A select statement creates deserialized data(columns) that is understood by Hive. An insert statement creates serialized data(files) that can be stored into an external storage like HDFS”.

Hive Row format and Map-reduce Input/Output format

In any table definition, there are two important sections.

The “Row Format” describes the libraries used to convert a given row into columns. The “Stored as” describes the Input Format and Output Format libraries used by map-reduce to read and write to HDFS files.

1. **Explain how Hive Deserializes and serialises the data?**

Hive, a data warehouse infrastructure built on top of Apache Hadoop, provides a SQL-like interface for querying and analyzing large datasets stored in Hadoop Distributed File System (HDFS) or other compatible file systems. When processing data in Hive, serialization and deserialization play a crucial role in converting data between its binary representation and a structured format that can be processed by Hive.

Hive uses a serialization/deserialization (SerDe) framework to handle the conversion of data between binary formats and structured representations. The SerDe framework defines two main components: the Serializer and the Deserializer.

1. Serializer: The Serializer is responsible for converting structured data into a binary format. It takes data objects in Hive's internal representation, such as rows or columns, and transforms them into a binary format suitable for storage or transmission. The serialized data can be written to files in HDFS or sent over the network.

2. Deserializer: The Deserializer performs the opposite operation of the Serializer. It takes binary data and converts it back into Hive's internal representation, allowing Hive to process and query the data. The Deserializer reads the serialized data from files or network streams and reconstructs the structured data objects that Hive can work with.

Hive supports various serialization formats, including:

- Text: The Text format represents data as plain text, with records separated by newlines and fields separated by delimiters (e.g., commas or tabs). It is human-readable but less space-efficient compared to binary formats.

- SequenceFile: The Sequence File format is a binary format optimized for storing large amounts of key-value pairs. It provides efficient compression and serialization of data.

- Avro: Avro is a data serialization system that provides a compact binary format along with a schema for data validation and compatibility. Hive can use Avro for both serialization and deserialization.

- ORC (Optimized Row Columnar): ORC is a columnar storage format specifically designed for Hive. It provides high compression and efficient column-wise access, making it suitable for analytical workloads.

- Parquet: Parquet is another columnar storage format

1. **Write the name of the built-in serde in hive.**

The built-in serde (Serializer/Deserializer) in Apache Hive is called "LazySimpleSerDe." It is a simple serde that is used by default in Hive for handling various data formats such as CSV (comma-separated values) and TSV (tab-separated values). The LazySimpleSerDe is designed to provide a balance between performance and flexibility for basic data serialization and deserialization tasks in Hive.

1. **What is the difference between LIKE and RLIKE operators in Hive?**

In Hive, LIKE and RLIKE are both operators used for pattern matching in queries, but they have some differences in their functionality.

**1. LIKE Operator:**

- The LIKE operator is used for simple pattern matching using wildcards.

- It matches a string value against a pattern and returns true if the pattern matches, and false otherwise.

- The pattern can contain two wildcard characters:

- '%' (percent sign): Matches any sequence of characters (including zero characters).

- '\_' (underscore): Matches any single character.

- For example, `column LIKE 'abc%'` will match any string that starts with "abc", such as "abc123" or "abcdef".

**2. RLIKE Operator:**

- The RLIKE (or REGEXP) operator is used for pattern matching using regular expressions.

- It matches a string value against a regular expression pattern and returns true if the pattern matches, and false otherwise.

- Regular expressions allow for more complex pattern matching and can express a wide range of patterns.

- RLIKE uses the Java regular expression syntax, which provides powerful pattern matching capabilities.

- For example, `column RLIKE '^ab[0-9]+$'` will match any string that starts with "ab" followed by one or more digits, such as "ab123" or "ab4567".

In summary, the LIKE operator is simpler and uses wildcard characters for pattern matching, while the RLIKE operator supports more advanced pattern matching using regular expressions. The choice between them depends on the complexity of the patterns you need to match in your queries.

1. **How to change the column data type in Hive?**

By using this command below one can change the column data type: ALTER TABLE table\_name

CHANGE column\_name column\_name new\_datatype; I hope this works.

1. **How will you convert the string ’51.2’ to a float value in the particular column?**

Select (colname)asFLOAT

1. **What will be the result when you cast ‘abc’ (string) as INT?**

When you attempt to cast the string 'abc' to an integer (INT), a ValueError will be raised. This is because 'abc' cannot be interpreted as a valid integer representation. In Python, the int() function expects a string that can be parsed into an integer, such as '123', '0', or '-42'. Since 'abc' does not meet this criterion, it will result in a ValueError.

1. **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;**

It creates partition on table employees with partition values coming from the columns in the select clause. It is called Dynamic partition insert.

1. **Write a query where you can overwrite data in a new table from the existing table.**

INSERT INTO TABLE tablename1

[PARTITION (partcol1=val1, partcol2=val2 ...)]

select\_statement1 FROM from\_statement;

1. **What is the maximum size of a string data type supported by Hive? Explain how Hive supports binary formats.**

The maximum size of a string data type supported by Hive is 2 GB. Hive supports the text file format by default, and it also supports the binary format sequence files, ORC files, Avro data files, and Parquet files.

* Sequence file: It is a splittable, compressible, and row-oriented file with a general binary format.
* ORC file: Optimized row columnar (ORC) format file is a record-columnar and column-oriented storage file. It divides the table in row split. Each split stores the value of the first row in the first column and follows subsequently.
* Avro data file: It is the same as a sequence file that is split table, compressible, and row-oriented but without the support of schema evolution and multilingual binding.
* Parquet file: In Parquet format, along with storing rows of data adjacent to one another, we can also store column values adjacent to each other such that both horizontally and vertically datasets are partitioned.

1. **What File Formats and Applications Does Hive Support?**

Hive facilitates managing large data sets supporting multiple data formats, including comma-separated value (. csv) TextFile, RCFile, ORC, and Parquet.

1. **How do ORC format tables help Hive to enhance its performance?**

You can easily store the Hive Data with the ORC (Optimized Row Column) format, which helps to streamline several limitations.

1. **How can Hive avoid mapreduce while processing the query?**

When queried SELECT, FILTER, LIMIT queries, this property skip mapreduce and using FETCH task. As a result, Hive can execute query without run mapreduce task.

1. **What is view and indexing in hive?**

Views are similar to tables, which are generated based on the requirements.

* We can save any result set data as a view in Hive
* Usage is similar to as views used in SQL
* All type of DML operations can be performed on a view

Creation of View:

Syntax:

Create VIEW <VIEWNAME> AS SELECT

Example:

Hive>Create VIEW Sample\_ViewAS SELECT \* FROM employees WHERE salary>25000

In this example, we are creating view Sample\_View where it will display all the row values with salary field greater than 25000.

***Index -***

Indexes are pointers to particular column name of a table.

* + The user has to manually define the index
  + Wherever we are creating index, it means that we are creating pointer to particular column name of table
  + Any Changes made to the column present in tables are stored using the index value created on the column name.

Syntax:

Create INDEX <INDEX\_NAME> ON TABLE < TABLE\_NAME(column names)>

Example:

Create INDEX sample\_Index ON TABLE guruhive\_internaltable(id)

Here we are creating index on table guruhive\_internaltable for column name id.

1. **Can the name of a view be the same as the name of a hive table?**

The name of a view must be unique, and it cannot be the same as any table or database or view's

name.

1. **What types of costs are associated in creating indexes on hive tables?**

Basically, there is a processing cost in arranging the values of the column on which index is created since Indexes occupies.

1. **Give the command to see the indexes on a table.**

The command you can use to see the indexes of a table is written below:

SHOW INDEX ON table\_name

This command will list all the indexes on any of the columns with this table\_name command.

1. **Explain the process to access subdirectories recursively in Hive queries.**

We can use following commands in Hive to recursively access sub-directories:

hive> Set mapred.input.dir.recursive=true;

hive> Set hive.mapred.supports.subdirectories=true;

Once above options are set to true, Hive will recursively access sub-directories of a directory in MapReduce.

1. **If you run a select \* query in Hive, why doesn't it run MapReduce**?

The hive.fetch.task.conversion property of Hive lowers the latency of MapReduce overhead, and in effect when executing queries such as SELECT, FILTER, LIMIT, etc. it skips the MapReduce function.

1. **What are the uses of Hive Explode?**

Hadoop Developers consider an array as their input and convert it into a separate table row. To convert complicated data types into desired table formats, Hive uses Explode.

1. **What is the available mechanism for connecting applications when we run Hive as a server?**

* Thrift Client: Using Thrift, we can call Hive commands from various programming languages, such as C++, PHP, Java, Python, and Ruby.
* JDBC Driver: JDBC Driver enables accessing data with JDBC support, by translating calls from an application into SQL and passing the SQL queries to the Hive engine.
* ODBC Driver: It implements the ODBC API standard for the Hive DBMS, enabling ODBC-compliant applications to interact seamlessly with Hive.

1. **Can the default location of a managed table be changed in Hive?**

Absolutely, by using the LOCATION keyword, we can change the default location of Managed tables while creating the managed table in Hive. However, to do so, the user needs to specify the storage path of the managed table as the value to the LOCATION keyword, that will help to change the default location of a managed table.

1. **What is the Hive Object Inspector function?**

Hive uses ObjectInspector to analyse the internal structure of the row object and also the structure of the individual columns. ObjectInspector provides a uniform way to access complex objects that can be stored in multiple formats in the memory, including:

•Instance of a Java class (Thrift or native Java)

•A standard Java object (we use java.util.List to represent Struct and Array, and use java.util.Map to represent Map)

•A lazily-initialized object (For example, a Struct of string fields stored in a single Java string object with starting offset for each field) A complex object can be represented by a pair of ObjectInspector and Java Object.

The ObjectInspector not only tells us the structure of the Object, but also gives us ways to access the internal fields inside the Object.

1. **What is UDF in Hive?**

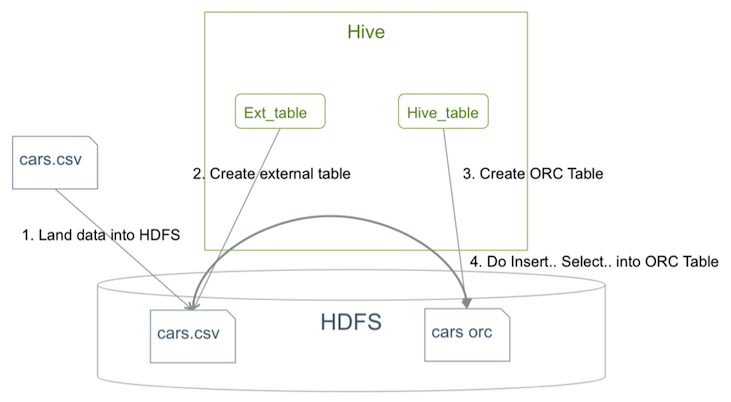
Basically, we can use two different interfaces for writing Apache Hive User Defined Functions.

Simple API

Complex API

As long as our function reads and returns primitive types, we can use the simple API (org.apache.hadoop.hive.ql.exec.UDF). In other words, it means basic Hadoop & Hive writable types. Such as Text, IntWritable, LongWritable, DoubleWritable, etc.

1. **Write a query to extract data from hdfs to hive.**



1. **What is TextInputFormat and SequenceFileInputFormat in hive.**

**TextInputFormat –**

TextInputFormat is the default input Format. Each record is a line of input. The key, a LongWritable, is the byte offset within the file of the beginning of the line. The value is the contents of the line, excluding any line terminators.

**SequenceFileInputFormat –**

Hadoop MapReduce is not restricted to processing textual data. It has support for binary formats, too. Hadoop’s sequence file format stores sequences of binary key-value pairs. Sequence files are well suited as a format for MapReduce data because they are splittable (they have sync points so that readers can synchronize with record boundaries from an arbitrary point in the file, such as the start of a split), they support compression as a part of the format, and they can store arbitrary types using a variety of serialization frameworks.

1. **How can you prevent a large job from running for a long time in a hive?**

This can be achieved by setting the MapReduce jobs to execute in strict mode set hive.mapred.mode=strict;

The strict mode ensures that the queries on partitioned tables cannot execute without defining a WHERE clause.

1. **When do we use explode in Hive?**

That is used to split a column, but instead of creating multiple rows, it creates multiple columns. This function is beneficial when working with maps. It allows us to split a map column into multiple columns, each containing one key-value pair from the map.

1. **Can Hive process any type of data formats? Why? Explain in very detail**

Hive supports four file formats those are TEXTFILE, SEQUENCEFILE, ORC and RCFILE (Record Columnar File).

For single user metadata storage, Hive uses derby database and for multiple user Metadata or shared Metadata case Hive uses MYSQL.

In summary, Hive is primarily designed for processing structured data with a well-defined schema. It can also handle semi-structured data by utilizing SerDe libraries, which enable it to read and write data in various formats. However, when dealing with unstructured data, Hive's capabilities are more limited, and specialized tools or preprocessing steps may be required to handle the complexity of the data structure.

1. **Whenever we run a Hive query, a new metastore\_db is created. Why?**

A local metastore is created when we run Hive in an embedded mode. Before creating, it checks whether the metastore exists or not, and this metastore property is defined in the configuration file, hive-site.xml.

The property is:

javax.jdo.option.ConnectionURL

with the default value:

jdbc:derby:;databaseName=metastore\_db;create=true

Therefore, we have to change the behaviour of the location to an absolute path so that from that location the metastore can be used.

1. **Can we change the data type of a column in a hive table? Write a complete query.**

ALTER TABLE table\_name CHANGE column\_name column\_name new\_datatype;

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?**

When using the `LOAD DATA` clause in Hive to load data into a table, you can specify that the file is located in HDFS (Hadoop Distributed File System) by providing the HDFS file path instead of a local file path.

To specify it is an HDFS file, you need to use the HDFS file path prefix `hdfs://` followed by the path to the file in HDFS. Here's an example:

```sql

LOAD DATA INPATH 'hdfs://<HDFS\_FILE\_PATH>' INTO TABLE <table\_name>;

```

Replace `<HDFS\_FILE\_PATH>` with the actual path to the file in HDFS, and `<table\_name>` with the name of the Hive table where you want to load the data.

For instance, if the file is located at `/user/data/file.txt` in HDFS, the `LOAD DATA` statement would look like this:

```sql

LOAD DATA INPATH 'hdfs:///user/data/file.txt' INTO TABLE <table\_name>;

```

Note the triple forward slashes (`///`) after `hdfs:` which indicate the HDFS file path.

By specifying the HDFS file path in the `LOAD DATA` statement, Hive will recognize that it needs to load data from an HDFS file rather than a local file.

1. **What is the precedence order in Hive configuration?**

In Hive we can use following precedence order to set the configurable properties.

Hive SET command has the highest priority

-hiveconf option from Hive Command Line

hive-site.xml file

hive-default.xml file

hadoop-site.xml file

hadoop-default.xml file

1. **Which interface is used for accessing the Hive metastore?**

WebHCat API web interface can be used for Hive commands. It is a REST API that allows applications to make HTTP requests to access the Hive metastore (HCatalog DDL). It also enables users to create and queue Hive queries and commands.

1. **Is it possible to compress json in Hive external table?**

Yes, you need to gzip your files and put them as is (\*.gz) into the table location.

1. **What is the difference between local and remote metastores?**

– Local Mode:

Basically, as the main HiveServer process, the Hive metastore service runs in the same process, but make sure Hive metastore database runs in a separate process, as well as it can be on a separate host, in Local mode.

– Remote Mode:

Whereas, the Hive metastore service runs in its own JVM process. Some processes(HiveServer2, HCatalog, Cloudera Impala™,) communicate with it with the help of the Thrift network API, in Remote mode.

In comparison with the Local mode, there is one benefit of using the Remote mode, that is Remote mode does not need the administrator to share JDBC login information for the metastore database along with each Hive user, but local mode does.

1. **What is the purpose of archiving tables in Hive?**

It only reduces the number of files which becomes easier for name node to manage.

1. **What is DBPROPERTY in Hive?**

In Hive, the `DBPROPERTY` function is used to retrieve the value of a specified property associated with a database. It allows you to access various metadata properties that are associated with a Hive database. The function syntax is as follows:

```

DBPROPERTY(database\_name, property\_name)

```

- `database\_name`: The name of the database for which you want to retrieve the property value.

- `property\_name`: The name of the property whose value you want to retrieve.

The `DBPROPERTY` function returns the value of the specified property as a string. It can be used in Hive queries to obtain information about the database properties.

For example, you can use the `DBPROPERTY` function to retrieve the location of a database:

```

SELECT DBPROPERTY('my\_database', 'location');

```

This query will return the location of the `'my\_database'` database as a string.

Note that the availability and specific properties supported by the `DBPROPERTY` function may vary depending on the version of Hive you are using and the configuration of your Hive environment.

1. **Differentiate between local mode and MapReduce mode in Hive.**

Both MapReduce mode and local mode seem same to the user but the difference is the way they execute.

**MapReduce mode:**

In MapReduce mode, Pig script is executed on Hadoop cluster. The Pig scripts are converted into MapReduce jobs and then executed on Hadoop cluster (hdfs)

**Local mode:**

In this mode, Pig script runs on a Single machine without the need of Hadoop cluster or hdfs. Local mode is used for development purpose to see how the script would behave in an actual environment.