**Hive Scenarios and realtime Questions**

**What are the uses of explode Hive?**

Array inputs can be converted into separate table rows. To convert complex data types into desired table formats Hive is essentially using explode.

**If you run a select \* query in Hive, Why does it not run MapReduce?**

The hive.fetch.task.conversion property of Hive lowers the latency of mapreduce overhead and in effect when executing queries like SELECT, FILTER, LIMIT, etc.,

it skips mapreduce function

**How to skip header/footer rows from a table in Hive?**

TBLPROPERTIES("skip.header.line.count"="2”);

TBLPROPERTIES("skip.footer.line.count"="1");

**The maximum size of string data type supported by hive is 2 GB.**

**By using below commands we can access sub directories recursively in Hive**

set mapred.input.dir.recursive=true;

set hive.mapred.supports.subdirectories=true;

**Hive to read fixed width data set:**

CREATE EXTERNAL TABLE customers (userid STRING, name STRING)

ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'

WITH SERDEPROPERTIES ("input.regex" = "(.{10})(.{10})" )

LOCATION '/user/hduser/custdata';

**How to set hive table not allow duplicate loading of same data set:**

An immutable table is protected against accidental updates due to a script loading data into it being run multiple times by mistake.

The first insert into an immutable table succeeds and successive inserts fail, resulting in only one set of data in the table, instead of silently succeeding with multiple copies of the data in the table.

A table can be made immutable by creating it with TBLPROPERTIES ("immutable"="true").

The default is "immutable"="false".

INSERT INTO behavior into an immutable table is disallowed if any data is already present, although INSERT INTO still works if the immutable table is empty.

The behavior of INSERT OVERWRITE is not affected by the "immutable" table property.

drop table customers1;

CREATE EXTERNAL TABLE customers1 (userid STRING, name STRING)

row format delimited fields terminated by ','

LOCATION '/user/hduser/custdata1'

TBLPROPERTIES ("immutable"="true");

**Maximum hive table performance using orc with snappy compression:**

Create table with orc and snappy compressed.

drop table customers2;

CREATE EXTERNAL taBLE customers2 (userid STRING, name STRING)

row format delimited fields terminated by ','

stored as orc

LOCATION '/user/hduser/custdata2'

TBLPROPERTIES ("orc.compress"="SNAPPY");

**Changing table from external to managed and vice versa.**

drop TABLE IF EXISTS customers3;

CREATE EXTERNAL taBLE customers3 (userid STRING, name STRING)

row format delimited fields terminated by ','

stored as orc

LOCATION '/user/hduser/custdata2'

TBLPROPERTIES ("orc.compress"="SNAPPY");

alter table customers3 SET TBLPROPERTIES('EXTERNAL'='FALSE');

**Common table expression (CTE):**

If you refer a same query multiple times then go for CTE that acts like a temperory table for that session.

Customers who are in the category of both Games and puzzles

with T1 as ( select custno,category from txnrecords where category='Games'),

T2 as ( select custno,category from txnrecords where category='Puzzles')

select \*

from T1 inner join T2

ON T1.custno=t2.custno;

**How can you add a new partition for the month january in the above partitioned table?**

For adding a new partition in the above table partitioned\_transaction, we will issue the command give below:

ALTER TABLE partitioned\_transaction ADD PARTITION (month='January') LOCATION ‘/partitioned\_transaction’;

**What is the default maximum dynamic partition that can be created by a mapper/reducer? How can you change it?**

By default the number of maximum partition that can be created by a mapper or reducer is set to 100. One can change it by issuing the following command:

SET hive.exec.max.dynamic.partitions.pernode = <value>

Note: You can set the total number of dynamic partitions that can be created by one statement by using:

SET hive.exec.max.dynamic.partitions = <value>

**Why do we need buckets?**

A map side join requires the data belonging to a unique join key to be present in the same partition. But what about those cases where your partition key differs from that of join key? Therefore, in these cases you can perform a map side join by bucketing the table using the join key.

Bucketing makes the sampling process more efficient and therefore, allows us to decrease the query time.

**How will you consume this CSV file into the Hive warehouse using built SerDe?**

SerDe stands for serializer/deserializer. A SerDe allows us to convert the unstructured bytes into a record that we can process using Hive. SerDes are implemented using Java.

Hive comes with several built-in SerDes and many other third-party SerDes are also available.

Hive provides a specific SerDe for working with CSV files. We can use this SerDe for the sample.csv by issuing following commands:

CREATE EXTERNAL TABLE sample

(id int, first\_name string,

last\_name string, email string,

gender string, ip\_address string)

ROW FORMAT SERDE ‘org.apache.hadoop.hive.serde2.OpenCSVSerde’

STORED AS TEXTFILE LOCATION ‘/temp’;

Now, we can perform any query on the table ‘sample’:

SELECT first\_name FROM sample WHERE gender = ‘male’;

**Managing smaller files in hive table:**

Suppose, I have a lot of small CSV files present in /input directory in HDFS and I want to create a single Hive table corresponding to these files. The data in these files are in the format: {id, name, e-mail, country}. Now, as we know, Hadoop performance degrades when we use lots of small files.

So, how will you solve this problem where we want to create a single Hive table for lots of small files without degrading the performance of the system?

One can use the SequenceFile format which will group these small files together to form a single sequence file. The steps that will be followed in doing so are as follows:

*Create a temporary table:*

CREATE TABLE temp\_table (id INT, name STRING, e-mail STRING, country STRING)

ROW FORMAT FIELDS DELIMITED TERMINATED BY ‘,’ STORED AS TEXTFILE;

*Load the data into temp\_table:*

LOAD DATA INPATH ‘/input’ INTO TABLE temp\_table;

*Create a table that will store data in SequenceFile format:*

CREATE TABLE sample\_seqfile (id INT, name STRING, e-mail STRING, country STRING)

ROW FORMAT FIELDS DELIMITED TERMINATED BY ‘,’ STORED AS SEQUENCEFILE;

*Transfer the data from the temporary table into the sample\_seqfile table:*

INSERT OVERWRITE TABLE sample SELECT \* FROM temp\_table;

Hence, a single SequenceFile is generated which contains the data present in all of the input files and therefore, the problem of having lots of small files is finally eliminated.

**ADDITIONAL INTERVIEW QUESTIONS**

**1.What is view? At first, we have created a view on top of the table (two rows- empid, empname). Then added 100 Rows on the table whether the newly added rows will see in view?**

A view is a virtual table and is created based on the result from one or more real table.

Here is the syntax to create view on the table:-

**CREATE VIEW view\_name AS SELECT col1, col2 FROM tablename;**

Then you can view the data of the view using the below command-

**Select \* from view\_name;**

If you later want to drop the view, just use the below statement-

**Drop view view\_name;**

Now as you know what view is and how to create a view, let’s jump to the second statement of the question.

As the view is created with the contents of the table itself and so updating table will update the view as well. So if you will add 100 new rows to the table then while checking the view, it will also have those newly added 100 rows.

**2.At first, we have created a view on top of the table (two rows empid, empname). Then again added a third row (i.e. address ). Whether can we see the newly added row in the view?**

Here you just need to ensure that while creating the view no such condition has been applied which will prevent the view from updating.

For example, if you have just used a couple of column names (col1 and col2) while creating the view then if you will add the third column in the table also, it won’t affect the view.

**3. Without loading the data from HDFS is it possible to load the data to hive?**

Yes, you can use the files placed on the local system. Another option is to copy the other tables data t create a new table. But in that case also, indirectly you are using the HDFS data.

So, the best option to load data in Hive without copying in HDFS is to use local inpath command while loading data in Hive table as below.

"Load data local inpath ‘localfilepath’ into table tbl\_name"

**4.In HDFS the data will store why again we need to load the data to HIVE?**

Actually, Hive does not store anything. It is the HDFS which store the data which we put into Hive table.

The main reason to load the data again in Hive table is for further computation. For example, you have partitioned table, and so the system will make directories based on the partition and will keep file related to that partition.

Now if you will manually place the file, that won’t be accessible for any operation. So to avoid such conditions, we load the data in Hive table even we have files in HDFS location.

**5. Is multiline comment supported in Hive Script ?**

No.

**6. Does Hive support record level Insert, delete or update?**

Hive does not provide record-level update, insert, or delete by default, but supports when the table is enabled with transactional true and in orc format.

**7. What kind of datawarehouse application is suitable for Hive?**

Hive is not a full database. The design constraints and limitations of Hadoop and HDFS impose limits on what Hive can do.

**8.Hive is most suited for data warehouse applications, where**

1) Relatively static data is analyzed,

2) Fast response times are not required, and

3) When the data is not changing rapidly.

Hive doesn’t provide crucial features required for OLTP, Online Transaction Processing. It’s closer to being an OLAP tool, Online Analytic Processing.So, Hive is best suited for data warehouse applications, where a large data set is maintained and mined for insights, reports, etc.

**9. How can the columns of a table in hive be written to a file?**

By using awk command in shell, the output from HiveQL (Describe) can be written to a file.

hive -S -e "describe table\_name;" | awk -F" " ’{print 1}’ > ~/output.

**10. CONCAT function in Hive with Example?**

CONCAT function will concat the input strings. You can specify any number of strings separated by comma.

**Example:**

CONCAT ('Hive','-','performs','-','good','-','in','-','Hadoop');

**Output:**

Hive-performs-good-in-Hadoop

So, every time you delimit the strings by '-'. If it is common for all the strings, then Hive provides another command CONCAT\_WS. Here you have to specify the delimit operator first.

CONCAT\_WS ('-','Hive','performs','good','in','Hadoop');

**Output:** Hive-performs-good-in-Hadoop

**11.REPEAT function in Hive with example?**

REPEAT function will repeat the input string n times specified in the command.

**Example:**

REPEAT('Hadoop',3);

**Output:**

HadoopHadoopHadoop.

Note: You can add a space with the input string also.

**12.TRIM function in Hive with example?**

TRIM function will remove the spaces associated with a string.

**Example:**

TRIM(' Hadoop ');

**Output:** Hadoop.

Note: If you want to remove only leading or trialing spaces then you can specify the below commands respectively.

LTRIM(' Hadoop');

RTRIM('Hadoop ');

**13.REVERSE function in Hive with example?**

REVERSE function will reverse the characters in a string.

**Example:**

REVERSE('Hadoop');

**Output:**

poodaH

**14.LOWER or LCASE function in Hive with example?**

LOWER or LCASE function will convert the input string to lower case characters.

**Example:**

LOWER('Hadoop');

LCASE('Hadoop');

**Output:** hadoop

Note:

If the characters are already in lower case then they will be preserved.

**15.UPPER or UCASE function in Hive with example?**

UPPER or UCASE function will convert the input string to upper case characters.

Example:

UPPER('Hadoop');

UCASE('Hadoop');

Output: HADOOP

Note:

If the characters are already in upper case then they will be preserved.

**16.Double type in Hive – Important points?**

It is important to know about the double type in Hive. Double type in Hive will present the data differently unlike RDBMS.

See the double type data below:

24624.0

32556.0

3.99893E5

4366.0

E5 represents 10^5 here. So, the value 3.99893E5 represents 399893. All the calculations will be accurately performed using double type. The maximum value for a IEEE 754 double is about 2.22E308.

It is crucial while exporting the double type data to any RDBMS since the type may be wrongly interpreted. So, it is advised to cast the double type into appropriate type before exporting.

**17. Rename a table in Hive – How to do it?**

Using ALTER command, we can rename a table in Hive.

ALTER TABLE hive\_table\_name RENAME TO new\_name;

There is another way to rename a table in Hive. Sometimes, ALTER may take more time if the underlying table has more partitions/functions. In that case, Import and export options can be utilized. Here you are saving the hive data into HDFS and importing back to new table like below.

EXPORT TABLE tbl\_name TO 'HDFS\_location';

IMPORT TABLE new\_tbl\_name FROM 'HDFS\_location';

If you prefer to just preserve the data, you can create a new table from old table like below.

CREATE TABLE new\_tbl\_name AS SELECT \* FROM old\_tbl\_name;

DROP TABLE old\_tbl\_name;

**18. How to change a column data type in Hive?**

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

Example: If you want to change the data type of ID column from integer to bigint in a table called employee.

ALTER TABLE employee CHANGE id id BIGINT;

**19. Difference between order by and sort by in hive?**

SORT BY will sort the data within each reducer. You can use any number of reducers for SORT BY operation.

ORDER BY will sort all of the data together, which has to pass through one reducer. Thus, ORDER BY in hive uses single reducer.

ORDER BY guarantees total order in the output while SORT BY only guarantees ordering of the rows within a reducer. If there is more than one reducer, SORT BY may give partially ordered final results

**20. RLIKE in Hive?**

RLIKE (Right-Like) is a special function in Hive where if any substring of A matches with B then it evaluates to true. It also obeys Java regular expression pattern. Users don't need to put % symbol for a simple match in RLIKE.

Examples:

'Express' RLIKE 'Exp' --> True

'Express' RLIKE '^E.\*' --> True (Regular expression)

Moreover, RLIKE will come handy when the string has some spaces. Without using TRIM function, RLIKE satisfies the required scenario. Suppose if A has value 'Express ' (2 spaces additionally) and B has value 'Express' RLIKE will work better without using TRIM.

'Express' RLIKE 'Express' --> True

**Note:**

RLIKE evaluates to NULL if A or B is NULL.

**21. How to optimize Hive Performance?**

**Technique #1: Use Tez**

Hive can use the Apache Tez execution engine instead of the venerable Map-reduce engine. I won’t go into details about the many benefits of using Tez which are mentioned here; instead, I want to make a simple recommendation: if it’s not turned on by default in your environment, use Tez by setting to ‘true’ the following in the beginning of your Hive query:

**set hive.execution.engine=tez;**

With the above setting, every HIVE query you execute will take advantage of Tez.

**Technique #2: Use ORCFile**

Hive supports ORCfile, a new table storage format that sports fantastic speed improvements through techniques like predicate push-down, compression and more.

Using ORCFile for every HIVE table should really be a no-brainer and extremely beneficial to get fast response times for your HIVE queries.

As an example, consider two large tables A and B (stored as text files, with some columns not all specified here), and a simple query like:

SELECT A.customerID, A.name, A.age, A.address join

B.role, B.department, B.salary

ON A.customerID=B.customerID;

This query may take a long time to execute since tables A and B are both stored as TEXT. Converting these tables to ORCFile format will usually reduce query time significantly:

CREATE TABLE A\_ORC (

customerID int, name string, age int, address string

) STORED AS ORC tblproperties (“orc.compress" = “SNAPPY”);

INSERT INTO TABLE A\_ORC SELECT \* FROM A;

CREATE TABLE B\_ORC (

customerID int, role string, salary float, department string

) STORED AS ORC tblproperties (“orc.compress" = “SNAPPY”);

INSERT INTO TABLE B\_ORC SELECT \* FROM B;

SELECT A\_ORC.customerID, A\_ORC.name,

A\_ORC.age, A\_ORC.address join

B\_ORC.role, B\_ORC.department, B\_ORC.salary

ON A\_ORC.customerID=B\_ORC.customerID;

ORC supports compressed storage (with ZLIB or as shown above with SNAPPY) but also uncompressed storage.

Converting base tables to ORC is often the responsibility of your ingest team, and it may take them some time to change the complete ingestion process due to other priorities. The benefits of ORCFile are so tangible that I often recommend a do-it-yourself approach as demonstrated above – convert A into A\_ORC and B into B\_ORC and do the join that way, so that you benefit from faster queries immediately, with no dependencies on other teams.

**Technique #3: Use Vectorization**

Vectorized query execution improves performance of operations like scans, aggregations, filters and joins, by performing them in batches of 1024 rows at once instead of single row each time.

Introduced in Hive 0.13, this feature significantly improves query execution time, and is easily enabled with two parameters settings:

set hive.vectorized.execution.enabled = true;

set hive.vectorized.execution.reduce.enabled = true;

**Technique #4: cost based query optimization**

Hive optimizes each query’s logical and physical execution plan before submitting for final execution. These optimizations are not based on the cost of the query – that is, until now.

A recent addition to Hive, Cost-based optimization, performs further optimizations based on query cost, resulting in potentially different decisions: how to order joins, which type of join to perform, degree of parallelism and others.

To use cost-based optimization (also known as CBO), set the following parameters at the beginning of your query:

set hive.cbo.enable=true;

set hive.compute.query.using.stats=true;

set hive.stats.fetch.column.stats=true;

set hive.stats.fetch.partition.stats=true;

Then, prepare the data for CBO by running Hive’s “analyze” command to collect various statistics on the tables for which we want to use CBO.

For example, in a table tweets we want to collect statistics about the table and about 2 columns: “sender” and “topic”:

analyze table tweets compute statistics;

analyze table tweets compute statistics for columns sender, topic;

With HIVE 0.14 (on HDP 2.2) the analyze command works much faster, and you don’t need to specify each column, so you can just issue:

analyze table tweets compute statistics for columns;

That’s it. Now executing a query using this table should result in a different execution plan that is faster because of the cost calculation and different execution plan created by Hive.

**22. How do you check the performance of a HIVE query?**

EXPLAIN [EXTENDED|DEPENDENCY|AUTHORIZATION] query

**23. Assume there is a array column in a table, how will you show the data in row format?**

hive> SELECT explode(x) AS y FROM arrays;

a

b

c

d

e

**24. Can you create data structure (columns and data type) of a new table which is similar to existing table?**

CREATE TABLE new\_table LIKE existing\_table;

**25. What is the difference between Drop table verses TRUNCATE?**

Dropping Tables

The DROP TABLE statement deletes the data and metadata for a table. In the case of external

tables, only the metadata is deleted; the data is left untouched.

If you want to delete all the data in a table but keep the table definition, use TRUNCATE TABLE.

For example:

TRUNCATE TABLE my\_table;

This doesn’t work for external tables; instead, use dfs -rmr (from the Hive shell) to

remove the external table directory directly.

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

This can be done with following query

SHOW PARTITIONS table\_name PARTITION

(partitioned\_column =’ partition\_value’)

**27. Since there's no IFNULL, ISNULL, or NVL function supported on Hive, how to convert NULL to 0?**

COALESCE(column, CAST(0 AS BIGINT)) or NVL(value, default value)

Returns default value

if value is null else returns value.

**28. What is the use of explode in Hive?**

Explode in Hive is used to convert complex data types into desired table formats. explode UDTF basically emits all the elements in an array into multiple rows.

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

SORT BY – Data is ordered at each of ‘N’ reducers where the reducers can have overlapping range of data.

ORDER BY- This is similar to the ORDER BY in SQL where total ordering of data takes place by passing it to a single reducer.

DISTRUBUTE BY – It is used to distribute the rows among the reducers. Rows that have the same distribute by columns will go to the same reducer.

CLUSTER BY- It is a combination of DISTRIBUTE BY and SORT BY where each of the N reducers gets non overlapping range of data which is then sorted by those ranges at the respective reducers.

**30. There are several cases in which Dynamic Partitioning can be used to improve performance of queries. However, incorrect use of dynamic partitioning can also have a detrimental effect.**

A very common mistake in dynamic partition case is the below query:

insert into table abc partition( ds ) select \* from src;

Above query is fine if you have very few partitions and number of mappers from src is very limited. However, the effect of above query is:

Side effects of above problem:

Huge number of small files per partition

Small files always cause issues in upstream jobs because split computation on 3.65 mill file is going to take time compared to few hundred files

Memory issues during dynamic insert - in this case each mapper might be running parallel writers and creating internal buffers so mappers might fail with OOM. Likely case in Orc and Parquet

The solution for the above mentioned problems is to use 'distribute by' to make sure insert happens in reduce side and use distribute key on the partition key so that each reducer writes to some different partition(s).

For example, in the above example I was writing into a daily partition table for a years worth of data.

Instead it could simply be this:

set mapred.reduce.tasks=365;

insert into table abc partition( ds ) select \* from src distribute by ds;

What if it is writing into 10 years worth of data? - in that case above query will be ok, because then 1 reducer will be writing into 10 partitions.

**hive export delimited data:**