Scenario Based questions:

Q1. Will the reducer work or not if you use "Limit 1" in any HiveQL query?

The reducer may or may not work when you use "Limit 1" in any HiveQL query, as it depends on the nature of the query and the data being processed. If the query is such that it requires a reducer to process the data, then the reducer will still be used, but it will only process one row. However, if the query can be processed entirely by the mapper, then the reducer will not be used at all. (It is depended on **size** of the data).

Q2. Suppose I have installed Apache Hive on top of my Hadoop cluster using default metastore configuration. Then, what will happen if we have multiple clients trying to access Hive at the same time?

If you have installed Apache Hive on top of your Hadoop cluster using default metastore configuration, then the metastore will use an embedded Derby database. When multiple clients try to access Hive at the same time, they will be competing for access to the same Derby database. The metastore will handle these requests concurrently. This can cause **performance issues** and may result in some queries being blocked or delayed. To avoid this problem, you can use an external metastore database such as MySQL or PostgreSQL, which can handle concurrent connections more efficiently.

Q3. Suppose, I create a table that contains details of all the transactions done by the customers: CREATE TABLE transaction_details (cust_id INT, amount FLOAT, month STRING, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

Now, after inserting 50,000 records in this table, I want to know the total revenue generated for each month. But Hive is taking too much time in processing this query. How will you solve this problem and list the steps that I will be taking in order to do so?

Alter the table to add **partitioning** on the **'month'** column:

ALTER TABLE transaction_details ADD PARTITION (month='January');

ALTER TABLE transaction_details ADD PARTITION (month='February');

ALTER TABLE transaction_details ADD PARTITION (month='March');

----same for other months-----

Load data into the table, ensuring that the data is loaded into the correct partition:

LOAD DATA LOCAL INPATH '/path/to/data' INTO TABLE transaction_details PARTITION (month='January');

Run a query to calculate the revenue for each month, using the partitioning to limit the data scanned:

SELECT month, sum(amount) AS total_revenue FROM transaction_details GROUP BY month;

Q4. How can you add a new partition for the month December in the above partitioned table?

To add a new partition for the month of December in the above partitioned table, you can use the following command:

ALTER TABLE transaction_details ADD PARTITION (month='December');

Q5. I am inserting data into a table based on partitions dynamically. But I received an error – FAILED ERROR IN SEMANTIC ANALYSIS: Dynamic partition strict mode requires at least one static partition column. How will you remove this error?

To remove this error, you need to disable dynamic partitioning strict mode. You can do this by setting the 'hive.exec.dynamic.partition.mode' property to 'nonstrict'. Here's how:

SET hive.exec.dynamic.partition.mode = nonstrict;

Q6. How will you consume a CSV file into the Hive warehouse using built-in SerDe

Create an **external table** in Hive with the appropriate column names and data types matching the CSV file. For example:

```
CREATE EXTERNAL TABLE sample_table (
  id INT,
  first_name STRING,
  last_name STRING,
  email STRING,
  gender STRING,
  ip_address STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
```

"quoteChar" = "\""

)

LOCATION '/temp';

LOAD DATA INPATH '/temp/sample.csv' INTO TABLE sample_table;

SELECT * FROM sample_table;

Q7. Suppose, I have a lot of small CSV files present in the 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?

ANS - Use partitioning: If possible, partition the data based on a relevant column, such as the "country" column in your case. Partitioning can further improve performance by allowing Hive to skip irrelevant data while querying. For example, you can partition the data by country, resulting in separate directories/files for each country in HDFS.

Q8. Is it possible to add 100 nodes when we already have 100 nodes in Hive? If yes, how?

Yes, we can add the nodes by following the below steps:

Step 1: Take a new system; create a new username and password

Step 2: Install SSH and with the master node setup SSH connections

Step 3: Add ssh public_rsa id key to the authorized keys file

Step 4: Add the new DataNode hostname, IP address, and other details in /etc/hosts slaves file.

Step 5: Start the DataNode on a new node

Step 6: Login to the new node.

Hive Practical questions:

Hive Join operations

Create a table named CUSTOMERS(ID | NAME | AGE | ADDRESS | SALARY)

Create a Second table ORDER(OID | DATE | CUSTOMER_ID | AMOUNT

)

Now perform different joins operations on top of these tables

(Inner JOIN, LEFT OUTER JOIN ,RIGHT OUTER JOIN ,FULL OUTER JOIN)

```
CREATE TABLE CUSTOMERS (
  ID INT,
  NAME STRING,
  AGE INT,
  ADDRESS STRING,
  SALARY FLOAT
);
CREATE TABLE ORDERS (
  OID INT,
  DATE DATE,
  CUSTOMER_ID INT,
  AMOUNT FLOAT
);
```

Inner join

```
SELECT *
FROM CUSTOMERS
INNER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Left outer join

```
SELECT *
FROM CUSTOMERS
LEFT OUTER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Right outer join

```
SELECT *
FROM CUSTOMERS
RIGHT OUTER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Full outer join

```
SELECT *
FROM CUSTOMERS
FULL OUTER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Loading the Data from the XML file

1. Create a hive table as per given schema in your dataset.

& 2. try to place a data into table location

```
CREATE TABLE assignment_data (
  Date DATE,
  Time TIMESTAMP,
  CO_GT INT,
  PT08_S1_CO INT,
  NMHC_GT INT,
  C6H6_GT INT,
  PT08_S2_NMHC INT,
  NOx_GT INT,
  PT08_S3_NOx INT,
  NO2_GT INT,
  PT08_S4_NO2 INT,
  PT08_S5_O3 INT,
  T INT,
```

```
  RH INT,
  AH INT
)
ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'
WITH SERDEPROPERTIES (
  "column.xpath.Date"="/row/Date/text()",
  "column.xpath.Time"="/row/Time/text()",
  "column.xpath.CO_GT"="/row/CO_GT/text()",
  "column.xpath.PT08_S1_CO"="/row/PT08_S1_CO/text()",
  "column.xpath.NMHC_GT"="/row/NMHC_GT/text()",
  "column.xpath.C6H6_GT"="/row/C6H6_GT/text()",
  "column.xpath.PT08_S2_NMHC"="/row/PT08_S2_NMHC/text()",
  "column.xpath.NOx_GT"="/row/NOx_GT/text()",
  "column.xpath.PT08_S3_NOx"="/row/PT08_S3_NOx/text()",
  "column.xpath.NO2_GT"="/row/NO2_GT/text()",
  "column.xpath.PT08_S4_NO2"="/row/PT08_S4_NO2/text()",
  "column.xpath.PT08_S5_O3"="/row/PT08_S5_O3/text()",
  "column.xpath.T"="/row/T/text()",
  "column.xpath.RH"="/row/RH/text()",
  "column.xpath.AH"="/row/AH/text()"
)
STORED AS INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
LOCATION '/all_data/assignment_data';
```

3. Perform a select operation .

```
SELECT * FROM assignment_data;
```

4. Fetch the result of the select operation in your local as a csv file .

```
INSERT OVERWRITE LOCAL DIRECTORY '/path/to/local/directory'
ROW FORMAT DELIMITED
```

FIELDS TERMINATED BY ','

SELECT * FROM assignment_data;

5. Perform group by operation .

SELECT Date, COUNT(*) as num_records

FROM assignment_data

GROUP BY Date;

7. Perform filter operation at least 5 kinds of filter examples .

SELECT * FROM assignment_data WHERE CO_GT = 2.0;

SELECT * FROM assignment_data WHERE DATE = '2004-03-12';

SELECT * FROM assignment_data WHERE C6H6_GT > 10 AND NOx_GT < 100;

SELECT * FROM assignment_data WHERE RH <= 50;

SELECT * FROM assignment_data WHERE T >= 20 AND T <= 25;

8. show and example of regex operation

SELECT * FROM assignment_data WHERE Date RLIKE '1/1/2020';

9. alter table operation

In Hive, the ALTER TABLE statement is used to modify the structure of an existing table. Here are some examples of the ALTER TABLE operation:

ALTER TABLE assignment_data RENAME COLUMN date TO date_new;

10 . drop table operation

The DROP TABLE command is used to remove an existing table and its metadata from the Hive metastore. When a table is dropped, all data associated with the table is also deleted. The syntax for dropping a table is as follows:

```
DROP TABLE IF EXISTS assignment_data;
```

12 . order by operation .

```
SELECT *
FROM assignemnt_data
ORDER BY time;
```

13 . where clause operations you have to perform .

```
SELECT *
FROM assignment_data
WHERE date = '1/1/2020';
```

14 . sorting operation you have to perform .

```
SELECT *
FROM assignemnt_data
ORDER BY time asc;
```

15 . distinct operation you have to perform .

```
SELECT DISTINCT date FROM assignment_data;
```

16 . like an operation you have to perform .

```
CREATE TABLE my_table (
  id INT,
  name STRING,
  age INT
)
```

```
ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;
```

17 . union operation you have to perform .

```
SELECT date, T FROM assignment_data

UNION

SELECT date, RH FROM assignment_data;
```

18 . table view operation you have to perform .

```
CREATE VIEW assignment_data_view AS

SELECT date, time, RH FROM assignment_data

WHERE RH > 10;
```

Hive operation with python

Create a python application that connects to the Hive database for extracting data, creating sub tables for data processing, drops temporary tables.fetch rows to python itself into a list of tuples and mimic the join or filter operations

```
from pyhive import hive

# Connect to Hive database

conn = hive.connect(host='localhost', port=10000, username='myuser')

# Create a cursor object

cur = conn.cursor()

# Extract data from a table

cur.execute('SELECT * FROM my_table')
```

```
rows = cur.fetchall()

# Create a sub-table for processing
cur.execute('CREATE TABLE my_subtable AS SELECT * FROM my_table WHERE column1 > 10')

# Drop temporary table
cur.execute('DROP TABLE my_subtable')

# Fetch rows to Python as a list of tuples
cur.execute('SELECT * FROM my_table')
rows = cur.fetchall()

# Join and filter operations
filtered_rows = [row for row in rows if row[1] == 'value']
joined_rows = [(row1[0], row2[1]) for row1 in rows1 for row2 in rows2 if row1[0] == row2[0]]

# Close cursor and connection
cur.close()
conn.close()
```