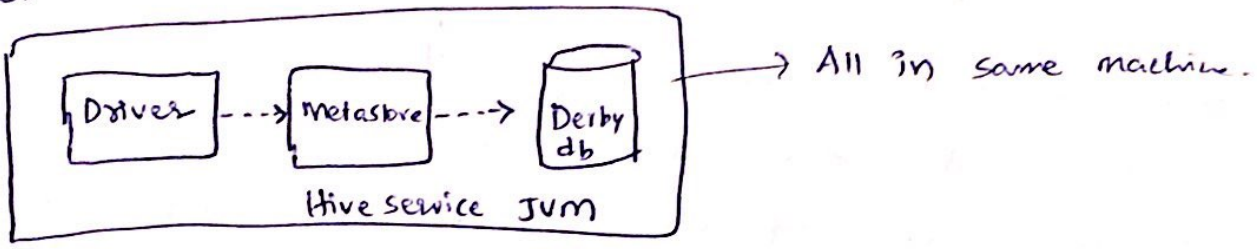
First Name: Sukesh   700 no: 700689100

Last Name: Armoor

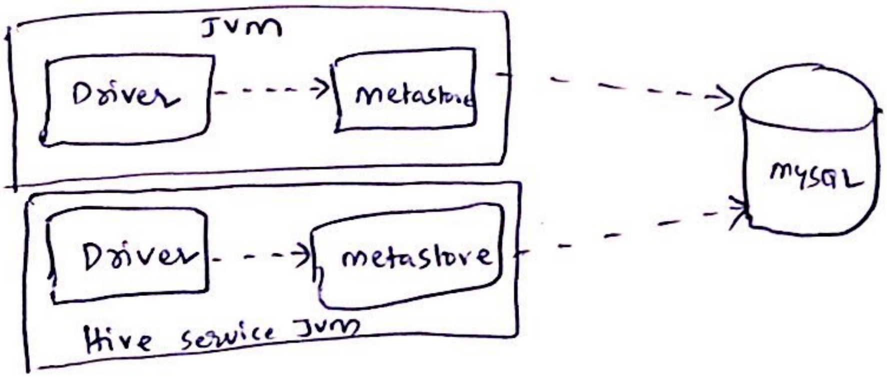
Question 1: The three configuration modes for running Hive CLI Service with respect to metastore service and meta-store database are: local, embeeded and remote.

* By default hive runs in cli mode.

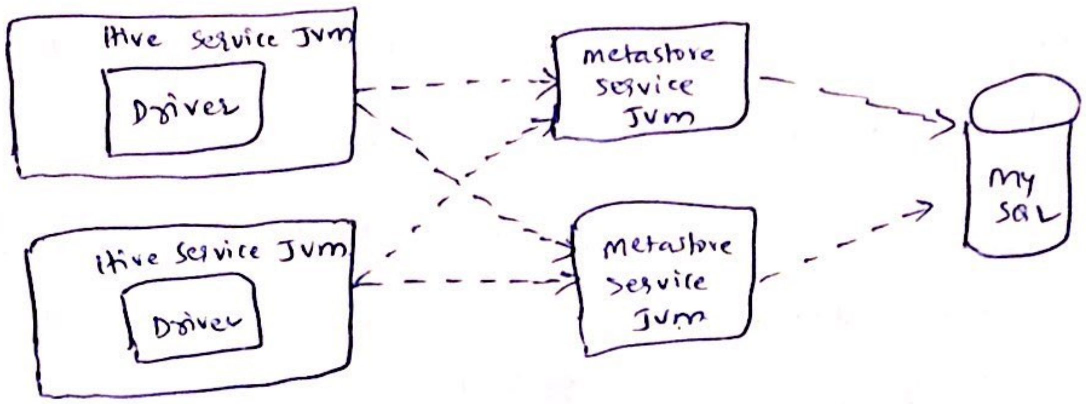
i) Local mode: In this mode, the metastore service runs in the same JVM as the hive service. The database i,e Java Derby is also embeeded with metastore service and driver. But only one user session can run at a time.



ii) Embeeded mode: Local mode supports only one users session at a time. Solution for overcoming this is to run hive session in embeeded mode. Here we use the standalone database i,e it will reside outside on other machine.



Remote: It is the best option to run metastore.
At client side, we need to run a CLI (or) hive server.
On other machine we can launch multiple sessions and run metastore services to allow large number of users.
· The metastore database resides on other machine.



Question 2: False.

* Hive is not required to install on every node in the cluster. As hive is a just a client for hadoop and it runs on top of hadoop, we don't have to install hive on every machine in the cluster.

8) (62)

Question 3: Give the file "transactions.txt" is stored under user home directory. we will populate the data into a table called Count.

> Create table Count (x String) Location '/user/hive/warehouse/transactions.txt';

Now the table Count will have all contents of transactions file.

we will split each line using the line delimeter i,e '\n'.

split ($\overset{x}{\cancel{\text{line}}}$, '\n') => It will split lines into array of strings.

we will select the length of each line using the following command.

> Select { split(length($\overset{x}{\cancel{\text{line}}}$), '\n') from Count;

The above command will display length of characters of each line.

To find total length of all lines we use sum

> Select split(sum (length (x), '\n')) from Count;

(4)

Question 4

4.a) (default) > Show databases;

4.b) (default)> Show tables;

4.c) (default)> Show tables in Companydb;

4.d) <default)> use company db;
<Companydb> Select * from employees;

4.e) <default)> use Company db;
(Companydb)> Select * from products limit 5;

4.f) Set hive.execution.engine;

4.g) Set hive.metastore.warehouse.dir;

4.h) (default)> !hdfs dfs -ls /user; (or) (default)> dfs -ls /user;

4.i) ! pwd;

4.j) Load data local inpath 'foo.txt' into table mytable;

4.k) (default)> describe formatted mytable; → display formatted o/p

(or)

(default)> describe extended mytable; → o/p is not formatted.

Question 5: Split function will split the given string and returns an array of strings. It will be in same row.

The explode function on split, will split the array of strings into multiple rows based on given delimeter (or) regular expression pattern.

```
explode( split ( 'Welcome to Programming Hive!', ' ' ));
```

o/p:
```
Welcome
to
programming
Hive!
```

Question 6: Hive enforces schema on read. Usually, in traditional database systems before doing update statements (or) writting output of a query (or) loading data, we have to verify the table schema and enter values accordingly. If not we will get an error. This is called 'schema on write'.

However, hive doesn't enforce schema on write. If we forget to enter any values, we will not get any error. Hive automatically displays NULL values. Hive will enforce constraint on 'Schema on read'.

~~Hence~~ Here are some differences

Schema on Read :	Schema on Write
i) Hive has no Control over the underlying Storage	i) Here the database has total control over the storage.
ii) Query results are slow	ii) Faster query results
iii) Hive tries its bes to recover from all the errors.	iii) We will encounter errors for any mistakes in updates
iv) It is unstructured.	iv) It is structured schema
v) It will fill the ~~gaps~~ records with null values if they are not specified	v) It will return error for any missing values

(04)

Question 7: ~~For~~ ~~Ex~~ If a table is declared external, dropping the table doesn't delete the data. Only meta data will be deleted. we can declare a table as external using the keyword 'EXTERNAL'.

```
Create external table if not exists mytable( id int,
          name    string,
     address ~~location~~  string )

              row format delimited
              fields terminated by ','
              location '/data/dataset-2020' ;
```

Questions (05)

8.a)
```
create table customers(
        cust_id    int,
        cust_name  string,
        street     string,
        city       string,
        zip        int,
        region     string)

        Partitioned by (country string)
        location '/data/customers';
```

8.b)
```
    from customers c
      insert ~~into~~ overwrite directory '/data/customers/usa'
       select * where c.country = 'usa'
      insert overwrite directory '/data/customers/canada'
         select * where c.country = 'canada'
      insert overwrite directory '/data/customers/mexico'
         select * where c.country = 'mexico';
```

8.c) We don't have to use dynamic partitioning in this
scenario. Here in the employees table already we
have ~~[illegible]~~ partitioned on the Country field. We
have only 3 values for Country. Partitioning is useful
when we have to scan through large amounts of
data.

* Dynamic partitioning is useful if you want to partition
numbers of columns but you don't know how many
columns to partition. But here only one country
column has partition.

8.d) Usually partitioning will minimize the table scan.
we use partitioning for faster query results. In the
given question partitioning is done on Country. Partition
tables will change how hive structures data storage.
Hive will create subdirectories for all the partitions
created. So here for each Country usa, canada and
mexica three partition directories will be created. Wheneve
we want to query any result for Country usa,
~~[illegible]~~ the query will refer to the subdirectory
So it don't have to scan entire data.

Thus we get faster query results.

06

Q.9)
```
X = sc. textFile ("hdfs://data/logfiles")

X. count ()
```

Q.10)
```
Lines = X. filter( lambda y: "error" in y):

    Lines. Collect ()
```

Q.11) X. map (lambda ~~y~~ z: len (z)). Sum()

Q.12)
```
x = range (0,100)
y = list (x)
z = sc. parallelize (y)
 . z. Sum ()
```

Q13) Lazy evaluation in spark:

lazy evaluation in spark means, whenever we call a transformation on an RDD, the operation is not performed immediately. It doesnot allocate memory for any data sets until they are computed. This feature of Spark will leave the memory unoccupied. Spark uses lazy evaluation to reduce number of passes it takes ~~to~~ to take over our data by grouping operations together.

For example, • logs = sc. textFile ("hello.txt") this command will not use any memory. Whenever we does any operation, then only memory is created and destroyed after the result.

• logs. Count () → uses memory.

~~Ex~~. Lazy evaluation applies only to actions. It doesn't apply to transformations.

Ex: action: logs. Count ()

transformation: logs = sc. textFile ("file1.txt")