HIVE Metastore Modes :

1. **Embedded Metastore:** By default, the Metastore service runs in the same JVM with the hive service. In this case it uses embedded derby database stored on the local file-system. This mode of Hive has a limitation that only one session could be opened at a time as only one embedded Derby database can access the database files on disk.
2. **Local Metastore:** Being a data-warehousing framework, a single session for Hive is not preferred. To solve this limitation of Embedded Metastore, a support for Local Metastore was developed. A separate database service runs as a process on same or remote machine. The Metastore service still runs in the same JVM within hive service.
3. **Remote Metastore:** There is one more configuration where one or more Metastore servers run as separate processes. This allows multiple Hive Clients to connect to a remote service rather than starting a Metastore service in the same JVM. A Hive service is configured to use a remote Metastore by adding hive.metastore.uris property to Metastore server URIs. This property ho0lds a comma-separated list of Metastore services. By default, the Hive service will connect to the first URI mentioned in the property. In case of a connection failure, it'll randomly choose any of the Metastore and will try to reconnect.

Apache Thrift (Thrift Server): Scalable cross-platform, cross language service which quickly transfers huge amt of data across different OS & different Languages. Supports C++, Java, Python, Ruby, Perl,

Developed by FB & Open Sourced in 2007

Developed in C++

Light weight & Language independent

Supports data transportation & Serialization

**********Creating Table without location********************

create database dvs_nv17;

use dvs_nv17;

set hive.cli.print.current.db=true;

This will show the above database u have used.

create table emp_et(empno int, ename string, job string, sal double, comm float, deptno int) row format delimited fields terminated by ',';

Observation : A folder(with Table name) will be created in default warehouse path i.e., /user/hive/warehouse

load data local inpath '/home/training/dvs/emp.csv' into table emp;

Observation : File will copied to /user/hive/warehouse in to the table folder i.e.,

/user/hive/warehouse/dvs_nv17.db/emp

***********Creating Table with location********************

This is mainly to avoid data movement problem. This is useful mainly when there other users are using same data for different analysis.

create table emp_et1(empno int, ename string, job string, sal double, comm float, deptno int) row format delimited fields terminated by ',' location '/user/training/dvs_hdfs/emp_dir';

Points to be kept in mind:

1. Location should be HDFS path & not LFS

2. Location should be a folder & not a file

Observation :

Since we gave the location of the direction in table creation statement, no directory will be created under default path. Instead it will make use of the directory which we gave in the location.

In this case, loading is not required since we gave the path of the existing directory in HDFS. There by no data movement is avoided. Data resides in the same folder untouched.

************** Dropping tables **************

drop table emp;

Whenever a table is dropped in HIVE,

1. It drops table metadata from Metastore DB(MYSQL)

2. It drops table folder in HDFS

Whether we create a hive table with location or without location, it drops both metadata from Metastore DB & Actual data folder from HDFS. To avoid this data deletion problem, we have external tables which we will discuss tmrw.

************** Dropping Database **************

Drop database dvs_nov17;

It drops if the database is not having any tables associated. If the database is having any tables in it, it throws an error.

In that case, we need to cascade option as below:

Drop database dvs_nv17 cascade;

This will drop database information from metastore DB & Database directory & all Table directories in HDFS associated with the Database.

****************** TYPES of TABLES**********************

Managed or Internal Tables:

All these tables are managed by hive under default path '/user/hive/warehouse'. This path can be changed in hive-site.xml but not advised very frequently. If these tables are dropped, both the meta data & actual data from hdfs gets deleted.

hive>create table emp(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',';

create table dept(deptno int, dname string, dloc string) row format delimited fields terminated by ',';

As a first step, all the metadata will be entered in Meta store DB(MYSQL) & an empty directory will be created under /user/hive/warehouse in hdfs. If the table is dropped, both meta data & actual data will be deleted permanently.

Use Case for Managed Tables:

- When we are sure that we don't want data after deletion
- When you want data to be temporary like the data that we use for testing purpose
- When Hive wants to Manage the table data completely not allowing any external source to use the table like PIG, MR, SQOOP etc.,

************Managed Table without location**********************

create table emp_et(empno int, ename string, job string, sal double, comm float, deptno int) row format delimited fields terminated by ',';

Observation : A copy of the file will be created in default warehouse path.

************Managed Table with location**********************

create table emp_et1(empno int, ename string, job string, sal double, comm float, deptno int) row format delimited fields terminated by ',' location '/user/training/dvs_hdfs/emp_dir';

Observation : No copy of the file is created since we are giving location keyword. ALso there is no file movement.

In either case whether it is with or without location, if a table is dropped, both metadata + actual data will be deleted.

External Tables:

These are the tables which we normally use in real time where all the data is loaded already & we just create schema on top of this & use for our reporting purpose.

In this case even if the table is dropped, the metadata is deleted from Meta store DB & actual data will still reside in the hdfs in the same path without any file movement.

hive>create external table emp_et1(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' location '/user/training/dvs_hdfs/emp_dir';

Here emp_dir is the directory which is already existing with 'n' no:of files inside.

We can also create a external table without giving location. The table directory will be created under /user/hive/warehouse with the table name as the directory name. It is very similar to the managed table but the only difference is this directory will not be deleted even if the table is dropped.

<u>Use Case for External Tables:</u>

- When data in hdfs is used by other applications like MR, PIG, SQOOP etc.,
- Mostly in real time, we go with external tables so that even if the table is dropped, data exists & metadata associated gets deleted

******************* PARTITIONS **********************

Static partitions:

In case of static partitions, we need to have separate files for each partition. While creating tables, partition columns should be mentioned at the end in partitioned by clause. Every time we load, we need to specify the partition value and file name.

hive>create table std(sname string, sid int) partitioned by (yop int)row format delimited fields terminated by ',';

hive>load data local inpath '/home/training/dvs/std_2011.csv' into table std partition(yop='2011');

hive>load data local inpath '/home/training/dvs/std_2012.csv' into table st partition(yop='2012');

hive>load data local inpath '/home/training/dvs/std_2013.csv' into table st partition(yop='2013');

Drawbacks with Static Partitions:

1. If no:of files are more, the no:of load commands we need to write will be more.

2. User should be careful while specifying the partition value & file name. Else wrong data will be loaded to the partitions.

3. If we have mix of all partitions data in one file, static partitions will not work.

Dynamic Partitions:

Steps involved in Creating Dynamic Partition Table:

1. Create Temporary (Non-partitioned) Table
2. Load data to temporary Table
3. Create Partition Table
4. Enable Dynamic Partitions
5. Insert data to partition table from temporary table
6. Drop temporary table

hive>create table std_temp1(sname string, sid int, year int) row format delimited fields terminated by ',';

hive>load data local inpath '/home/training/dvs/std/std_all.csv' into table std_temp1;

hive>set hive.exec.dynamic.partition.mode=nonstrict;

hive>create table std_dp(sname string, sid int) partitioned by (year int) row format delimited fields terminated by ',';

hive>insert into table std_dp partition(year) select * from std_temp;

hive>drop table std_temp;

Note: If there are any null values for year column, there will be a default hive partition created in warehouse directory under the table directory

Nested partitions:

create table std_temp1(sname string, sid int, year int, month int, day int) row format delimited fields terminated by ',';

load data local inpath '/home/training/dvs/std/ std_all_data.csv' into table std_temp2;

hive>create table std_dp1(sname string, sid int) partitioned by (year int, month int, day int) row format delimited fields terminated by ',';

insert into table std_dp1 partition(year,month,day) select * from std_temp2;

Can follow the same process as mentioned above by creating a temp table & inserting.

What if Partition columns are not in sequence :

- Create temporary table with same column sequence as input file
- Load data to temporary table
- Create partition table in the sequence that we wanted like year, month & day
- Insert data from temp table to partition table with same column sequence as partition table column sequence like

create table std_temp3(sname string, sid int, day int, month int, year int) row format delimited fields terminated by ',';

load data local inpath '/home/training/dvs/std/stud.txt' into table std_temp3;

create table std_dp3(sname string, sid int) partitioned by (year int, month int, day int) row format delimited fields terminated by ',';

insert into std_dp3 partition(year,month,day) select sname,sid,year,month,day from std_temp3;

Partitions with ET(with loc):

Create external table std_et3(sname string, sid int) partitioned by (year int) row format delimited fields terminated by ',' location '/user/training/dvs_hdfs/std';

Adding partitions :

alter table std_dp add partition (yop=2016);

alter table std_dp add partition (yop=2016,mop=02);

alter table std_dp add partition (yop=2016,mop=07,dop=31);

Dropping partitions :

hive>alter table std_dp drop partition (yop=2014);

hive>alter table std_dp drop partition (yop=2016,mop=02,dop=31); For Nested partitions

Renaming Partitions :

hive>alter table std_dp partition (yop=2016) rename to partition(yop=2015);

Exchanging partitions:

Exchanging partitions of one table to another table(structure should be same for both the tables):

hive>alter table std1 exchange partition(yop=2015) with table std;

hive> alter table std1 exchange partition(yop=2016) with table std;

Repairing or recovering partitions:

hive>msck repair table std_dp;   --may not work in the current version of hive you are using. Feature included from hive-0.14.0

Enabling NO DROP/OFF LINE on Partition:

Alter table std partition(year=2015) ENABLE NO_DROP; /* It will not allow u to drop this partition*/
Alter table std partition(year=2015) DISABLE NO_DROP; /* It will allow u to drop this partition*/

Alter table std partition(year=2015) ENABLE OFFLINE;   /* Hide the partition */

Alter table std partition(year=2015) DISABLE OFFLINE;   /* Un Hide the partition */

Writing linux or hdfs commands from hive shell

hive>!ls;

hive>!hadoop fs -ls;

Writing hive commands from Linux shell

hive -e "show databases";

hive -e "select * from dvs_oct.std_dp">sample.csv   --select statement output will be written to sample.csv into the current location.

hive -e "select * from dvs_oct.std_dp where yop=2014">>sample.csv --appending the same file with more data

Writing HIVE script

cat>sample.hql

create table std_temp (name string, id1 int, yop int) row format delimited fields terminated by ',';

load data local inpath '/home/training/dvs/std_det' into table std_temp;

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

create table std_par(name string, id1 int) partitioned by (yop int) row format delimited fields terminated by ',';

insert into table std_par partition(yop) select * from std_temp;

drop table std_temp;

Running hive script:

hive -f sample.hql

hive -f /home/training/hivescript.hql

hive –S -f /home/training/hivescript.hql

S – Suppress the Map Reduce Log

Joins

1. EQUI or INNER Join

select A.ename,A.empno, B.dname, B.dloc from emp_et1 A right outer join dept B ON (A.deptno=B.dno);

select A.* from emp A join dept B ON (A.deptno=B.deptno);

2. Left outer Join

select A.ename, A.sal, B.dname, B.dloc from emp A left outer join dept B ON (A.deptno=B.deptno);

3. Right outer Join

select A.ename, A.sal, B.dname, B.dloc from emp A right outer join dept B ON (A.deptno=B.deptno);

4. Full outer Join

select A.ename, A.sal, B.dname, B.dloc from emp A right outer join dept B ON (A.deptno=B.deptno);

5. Cross Join

select A.ename, A.sal, B.dname, B.dloc from emp A cross join dept B;

Views : Views are virtual tables which does not occupy any space. Views can be used to restrict the no:of columns & give read-only access to tables. A *view* allows a query to be saved and treated like a table.

Creating view on entire table(To give read only permissions) :

create view emp_view COMMENT 'View on emp table' as select * from emp;

Creating view on selected columns of the table:

create view dept_view as select deptno, dname from dept;

Creating view on joining 2 tables:

create view emp_dept_view as select A.empno, B.deptno, B.dname from emp A join dept B on A.deptno=B.deptno where B.dname='IT';

Creating view from a existing view :

create view emp_view1 as select * from emp_view;

Dropping a view

Drop view emp_view;

Variable Substitution(Parameter passing):

```
set hive.variable.substitute=true;
```

Passing parameters for single line commands thru Linux

```
tb=emp;

c1=empno;

c2=ename;

no=4;

hive -e "select $c1, $c2 from $tb limit $no";
```

Passing parameters for single line commands thru hive shell

```
Hive> set a=emp;

Hive> set b=dept;

Hive> set c1=empno;

Hive> select * from ${hiveconf:a};

Hive> select ${hiveconf:c1} from ${hiveconf:a};
```

Passing parameters for Script(hql file):

```
select ${hiveconf:c1}, ${hiveconf:c2} from ${hiveconf:tb} limit ${hiveconf:lm};

hive -hiveconf tb=emp -hiveconf c1=empno -hiveconf c2=ename -hiveconf lm=4 -f h.hql;

hive -S -hiveconf tablename=emp -hiveconf col1=empno -hiveconf col2=ename -hiveconf
limit_number=4 -f h.hql;
```

Declaring Variable & Value in Script(hql file):

```
Set db=dvs;

Set tb=emp;

Create database ${hiveconf:db};

Use ${hiveconf:db};

Create table ${hiveconf:tb}(empno int, ename string, job string);
```

Complex data types in HIVE

Sample data for working with complex datatypes has been given below. Also table creation, load & extract statements are also given.

ARRAY DATATYPE

cat>array.csv

Arun,2020,Maths$Physics$Chemistry,A

Sunil,3030,Biology$Physics$Chemistry,C

Pavan,4040,Civics$Economics$Commerce,B

hive>create table array_table(name string, id int, sub array<string>, grade string) row format delimited fields terminated by ',' collection items terminated by '$';

hive>load data local inpath '/home/training/dvs/complex/array_file.txt' into table array_table;

hive>select name, sub[1] from array_table;

MAP DATATYPE

cat>map.csv

123,SMITH,sal#5000$comm#500$bonus#100

345,ALLEN,sal#6000$comm#600$bonus#200

hive> create table map_table(empno int, ename string,pay map<string,int>) row format delimited fields terminated by ',' collection items terminated by '$' map keys terminated by '#';

hive>load data local inpath '/home/training/dvs/complex/map_file.txt' into table map_table;

hive>select empno, ename,pay["sal"] from map_table;

STRUCT DATATYPE

cat>struct.csv

123,Arun,Maths$Physics$Chemistry,fee#50000$concession#5000$scholarship#5000,Bangalore$Karnataka$560037

345,Allen,Biology$Physics$Chemistry,fee#60000$concession#6000$scholarship#4000,HydAP511010

hive>create table struct_table(sid int, sname string, sub array<string>, pay_details map<string,int>, addr struct<city:string, state:string, pin:int>) row format delimited fields terminated by ','

collection items terminated by '$' map keys terminated by '#';

hive> load data local inpath '/home_dir/a043049/tgt/structfile' into table struct_table;

hive> select * from struct_table;

123 Arun ["Maths","Physics","Chemistry"] {"fee":50000,"concession":5000,"scholarship":5000} {"city":"Bangalore","state":"Karnataka","pin":560037}

```
345    Allen  ["Biology","Physics"]  {"fee":60000,"concession":6000}
{"city":"Hyd","state":"AP","pin":null}
```

```
hive> select sub[1], pay["fee"],pay["scholarship"], addr.city from struct_table;
```

Indexes : Creating an index is common practice with relational databases when you want to speed access to a column or set of columns in your database. Without an index, the database system has to read all rows in the table to find the data you have selected. Indexes become even more essential when the tables grow extremely large, and as you now undoubtedly know, Hive thrives on large tables.

Deferred index builds can be very useful in workflows where one process creates the tables and indexes, another loads the data and builds the indexes and a final process performs data analysis.

Hive doesn't provide automatic index maintenance, so you need to rebuild the index if you overwrite or append data to the table. Also, Hive indexes support table partition.

```
Hive>CREATE INDEX emp_index ON TABLE emp(empno) AS 'COMPACT' WITH DEFERRED REBUILD;
```

COMPACT : This is Index type. It stores value & the address of the record in the file.

Note that the WITH DEFERRED REBUILD portion of the command prevents the index from immediately being built. It is an empty index.

To build the index you can issue the following command:

```
Hive>ALTER INDEX emp_index ON emp REBUILD;

Hive>show index on emp;

select * from emp where empno=123;

Hive>drop index emp_index on emp;
```

BITMAP : This is one more type of index which stores value & the list of occurrence of that particular value. Best suitable in case if the values are repetitive like True / False or Boolean values like 0 or 1 in the field being indexed.

```
Hive>CREATE INDEX emp_index ON TABLE emp (job) AS 'BITMAP' WITH DEFERRED REBUILD;

ANALYST hdfs://localhost:54310/user/hive/warehouse/dv1.db/emp1/emp.csv  [425,252]

CLERK   hdfs://localhost:54310/user/hive/warehouse/dv1.db/emp1/emp.csv  [0,393,459,360]

MANAGER hdfs://localhost:54310/user/hive/warehouse/dv1.db/emp1/emp.csv  [217,182,107]

PRESIDENT      hdfs://localhost:54310/user/hive/warehouse/dv1.db/emp1/emp.csv  [287]
```

Creating index on multiple columns.

CREATE INDEX emp_index1 ON TABLE emp (detpno,job) AS 'COMPACT' WITH DEFERRED REBUILD;

Index on partition tables

When table is dropped, index & Index tables automatically dropped.

BUCKETS

Bucketing is also one more way of decomposing a big data set into small & manageable data sets like partitioning.

Bucketing is mainly for 2 reasons:

1. For faster joins(Join Optimization)

2. Table Sampling

Creating a bucketed table

hive>create table emp_bk1(empno int, ename string, job string, sal float, comm float, deptno int) clustered by (deptno) sorted by (empno) into 3 buckets row format delimited fields terminated by ',';

create table emp_bk1(empno int, ename string, job string, sal float, comm float, deptno int) clustered by (deptno) into 3 buckets row format delimited fields terminated by ',';

Steps involved in creating a bucketed Table:

1. Create Temporary Table
2. Load data to temporary Table
3. Create Bucketed Table
4. Enable Bucketing
5. Insert data to bucketed table from temporary table
6. Drop temporary table

Enabling Bucketing

hive>set hive.enforce.bucketing=true;

hive>insert into table emp_bk select * from emp_et_wl;

Note : It creates index based on the bucketed column for faster lookups(Joins)

Getting bucket wise sampling:

hive>select * from emp_bk tablesample(1 out of 3 on deptno);

hive>select * from emp_bk tablesample(bucket 1 out of 3 on deptno);

hive>select * from emp_bk tablesample(1 out of 3 on deptno)

UNION

select * from emp_bk tablesample(3 out of 3 on deptno)

Getting percentage wise sampling:

hive>select * from emp_bk tablesample(10 percent);

FILE FORMATS

File formats are mainly for Fast retrieval, Faster Writing & compression.

TEXTFILE

RC

PARQUET

ORC

hive>create table emp(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' stored as textfile;

hive>load data local inpath '/home/training/dvs/emp.csv' into table emp;

hive>create table emp_rc(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' stored as rcfile;   (or)

hive>create table emp_rc stored as rcfile as select * from emp_text;

hive>insert into table emp_rc select * from emp;

Note : Try with other formats with some more data so that you can feel the difference the file formats. You can try all the file formats with latest version of cloudera.

hive>create table emp_orc(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' stored as orcfile;

hive>create table emp_seq(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' stored as sequencefile;

hive>create table emp_pr(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' stored as parquetfile;

hive>create table emp_av(empno int, ename string, job string, sal float, comm float, deptno int) row format delimited fields terminated by ',' stored as avrofile;

Go to warehouse directory and see the file formats which may not be in user understandable format.

Use cases:

a) If your data is delimited by some parameters then you can use TEXTFILE format.

b) If your data is in small files whose size is less than the block size then you can use SEQUENCEFILE format.

c) If you want to perform analytics on your data and you want to store your data efficiently for that then you can use RCFILE format.

d) If you want to store your data in an optimized way which lessens your storage and increases your performance then you can use ORCFILE format.

Miscellaneous:

Create a temporary table(supports from hive-0.14.0 onwards)

hive>Create tempoarary table std_temp(sname string, sid int, yop int) row format delimited fields terminated by ',';

The above table will have existence till the end of the session & will be deleted once we come out of the session.

Enabling auto purge

If trash is enabled the data will go to trash, in case of auto purge, the data will not go to trash and will be permanently deleted.

hive>set TBLPROPERTIES("auto.purge"="true");

Skipping 'n' lines of data while loading the data

hive>create table emp3(empno int, ename string, job string, sal double, comm float, deptno int) row format delimited fields terminated by ','
tblproperties('skip.header.line.count'='1','creator'='Rams','date'='2016-01-26',auto.purge=true);

By using this option, we use skip headers of the file while loading data to the table if the file is coming with column headers.

Creating Database with DB properties

hive>create database dvs11 comment 'DVS DB' with DBPROPERTIES('creator'='Rams','date'='2015-09-12');

Checking DB properties

Create database dvs location '/user/training/warehouse/dvs.db';

hive>describe database extended dvs11;

Creating table with Table properties

hive>create table emp(ename string, empno int) row format delimited fields terminated by ','
TBLPROPERTIES('creator'='Rams','date'='2015-09-12');

Checking Table properties

hive>describe formatted emp;

Switch a table from internal to external

hive>ALTER TABLE table_name SET TBLPROPERTIES('EXTERNAL'='TRUE');

Switch a table from external to internal

hive>ALTER TABLE table_name SET TBLPROPERTIES('EXTERNAL'='FALSE');

Changing from one data type to another data type

hive>select cast(sal as int) from emp;

Importing the structure of another table in hive

hive>create table emp3 as select * from empl where 1=2;

hive>create table t2 like emp;

Importing the structure of table & data of another table

hive>create table emp3 as select * from empl;

Truncating data in hive table without truncate command

hive>insert overwrite table emp1 select * from emp where 1=2;

hive>insert into table emp1 select * from emp;

Inserting data in to a hive table

hive>insert into table empl2 select * from empl;

What happens if warehouse path is changed ?

The default location of all the tables and databases will be changed & the existing databases & tables will still reside in the same location.

Restricting full table scan on partitioned tables:

Set hive.mapred.mode=strict;

This property does not allow to query partitioned tables without where class. This prevents very large job running for long time.

Hadoop job –list;

Hadoop job –kill job_id

What is the significance of 'IF EXISTS" clause while dropping a table?

When we issue the command DROP TABLE IF EXISTS table_name

Hive throws an error if the table being dropped does not exist.

Create table IF NOT EXISTS emp(empno int, ename string, job string);

Add columns to a table

hive>alter table empl2 add columns (bonus float);

Rename Table

hive>alter table emp3 rename to emp4;

Listing databases of similar pattern

hive>SHOW DATABASES LIKE 'dvs.*';

Listing tables of similar pattern

hive>SHOW TABLES LIKE 'e.*';

Displaying current DB

hive> set hive.cli.print.current.db=true;

Displaying column headers

hive>set hive.cli.print.header=true;

hive>CREATE TABLE test_change (a int, b int, c int);

Renaming a column (a to a1)

hive>ALTER TABLE test_change CHANGE a a1 INT;

Changing position of a column (a to a1 & its datatype to string & changing the position of the column)

hive>ALTER TABLE test_change CHANGE a1 a2 STRING AFTER b;

AFTER B means, position of a2 will be after b.

The new table's structure is: b int, a2 string, c int.

hive>ALTER TABLE test_change CHANGE c c1 INT FIRST;

Column c will be changed c1 & moved to first position.

The new table's structure is: c1 int, b int, a2 string.

MISC

What is skewed table & difference between Partition table & skewed table

When there are more partitions for a table & if the data in the partitions is unevenly distributed

like few partitions contain huge data and many partitions contain only few records, partitions will kill the performance wherein very few records are pushed to a partition. The more mappers will be invoked & more intermediate files will be created thereby increasing the no:of partitions. This is the concept of skewed table will come. In this concept, we can specify particular column values which are having huge data & make sure, the partitions are created only for these values & all the other data is pushed in to a single partition. In this case, no: partitions are reduced, no:of mappers are reduced, no:Of intermediate files are reduced.

Lets say I have table with 100 countries but single country having 90% of the data & 10% of the data belong to remaining 99 countries. In this case, if we go for normal partitions, it will kill the performance very badly because it has to traverse lot of directories & files inside which will waste time. The solution would be a skewed table where we can insist hive to store 90% (single country data) into a single partition and remaining 99 countries data into a single partition.

```
Hive> set hive.mapred.supports.subdirectories=true;
```

```
hive>create table emp_s(empno int, ename string, job string, sal double, comm float, deptno int, country string) skewed by (country) on ('US') stored as directories row format delimited fields terminated by ',';
```

```
hive>create table emp_s(empno int, ename char(20), job char(20), sal double, comm float, deptno int, country string) skewed by (country) on ('US','UK') stored as directories row format delimited fields terminated by ',';
```

Recover Partitions (MSCK REPAIR TABLE)

Hive stores a list of partitions for each table in its metastore. If, however, new partitions are directly added to HDFS (say by using hadoop fs -put command), the metastore (and hence Hive) will not be aware of these partitions unless the user runs ALTER TABLE table_name ADD PARTITION commands on each of the newly added partitions.

However, users can run a metastore check command with the repair table option:

```
hive>MSCK REPAIR TABLE table_name;
```

which will add metadata about partitions to the Hive metastore for partitions for which such metadata doesn't already exist. In other words, it will add any partitions that exist on HDFS but not in metastore to the metastore.

Multi Table insert

```
from emp
```

```
insert into table dept10 select * where deptno=10

insert into table dept20 select * where deptno=20;
```

Make sure the 2 tables are dept10 & dept20 already created.

There is one more feature available where we can also write the data of a single table in to multiple directories.

```
from emp

insert overwrite directory '/dvs_hdfs/mngr_data' select * where job='MANAGER'

insert overwrite directory '/dvs_hdfs/clerk_data' select * where job='CLERK';

hive.exec.max.dynamic.partitions.pernode=100

hive.exec.max.dynamic.partitions=1000
```

Hive drawbacks:

- Not designed for online transactional data processing.
- Does not offer real-time queries and row level updates.
- Latency for Hive queries is generally very high
- No unstructured data processing

Impala :

Use Case : Impala is well-suited to executing SQL queries for interactive exploratory analytics on large data sets. Hive and MapReduce are appropriate for very long running, batch-oriented tasks such as ETL.

Features:

1. Impala uses same Metastore as HIVE
2. Impala is developed by Cloudera
3. Impala is developed in C++ & MPP distributed processing
4. It uses in memory concept to process the data
5. It provides SQL interface on top of data stored in HDFS
6. This is interactive SQL & mainly for low latency queries.

Type impala-shell

Impala Server:

The Impala server is a distributed, massively parallel processing (MPP) database engine. It consists of different daemon processes that run on specific hosts within your CDH cluster.

1. **Impala Daemon(Implalad) :**

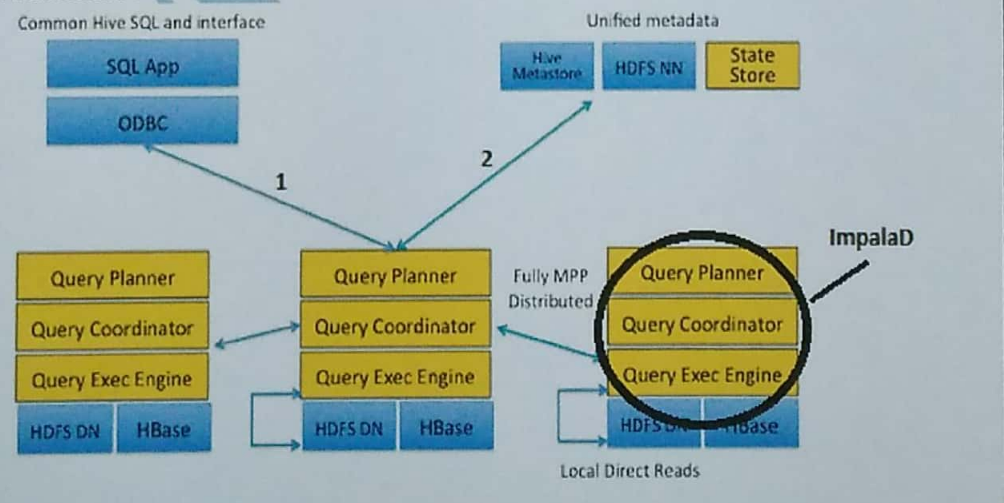
The core Impala component is a daemon process that runs on each DataNode of the cluster, physically represented by the impalad process. It reads and writes to data files; accepts queries transmitted from the impala-shell command, Hue, JDBC, or ODBC; parallelizes the queries and distributes work across the cluster; and transmits intermediate query results back to the central coordinator node. Handles client requests, handles query planning & query execution. You can submit a query to the Impala daemon running on any DataNode, and that instance of the daemon serves as the coordinator node for that query. The other nodes transmit partial results back to the coordinator, which constructs the final result set for a query.

2. Statestore(statestored): The Impala daemons are in constant communication with the statestore, to confirm which nodes are healthy and can accept new work. The Impala component known as the statestore checks on the health of Impala daemons on all the DataNodes in a cluster, and continuously relays its findings to each of those daemons. It is physically represented by a daemon process named statestored; you only need such a process on one host in the cluster. If an Impala daemon goes offline due to hardware failure, network error, software issue, or other reason, the statestore informs all the other Impala daemons so that future queries can avoid making requests to the unreachable node.

3. Catalog Server(catalogd):

The Impala component known as the catalog service relays the metadata changes from Impala SQL statements to all the DataNodes in a cluster. It is physically represented by a daemon process named catalogd; you only need such a process on one host in the cluster. They constantly send broadcast messages from the catalogd daemon (introduced in Impala 1.2) whenever any Impala node in the cluster creates, alters, or drops any type of object, or when an INSERT or LOAD DATAstatement is processed through Impala. This background communication minimizes the need for REFRESH or INVALIDATE METADATA statements that were needed to coordinate metadata across nodes

Architecture :



1. INVALIDATE METADATA for any new changes in metadata

2. refresh <Table Name> for any new file added to hive table

Comparision with HIVE

1. hive works on MR & Impala does not use MR
2. Hive has high-latency & Impala has low latency
3. Hive is much slow when compared to Impala. Depends on HW configuration.
4. Real time queries & adhoc queries

Drawbacks with Impala :

1. No indexing
2. No row level deletes
3. No Sampling
4. No Date Data type. Need to use string
5. No extended table/database

```
Hive>select count(*) from orders1;
```

```
Impala>select count(*) from orders1;
```

Hive Server2 & beeline:

- Hive CLI is legacy client while beeline is the new client that will replace Hive CLI.
- One of the main differences is beeline jdbc client connects to HS2 (Hive Server 2), while Hive CLI does not.
- HS2 provides security to hive but HS1 does not

Type beeline

```
Beeline>
```

```
!connect jdbc:hive2://localhost:10000 root cloudera
```

HCATALOG:

1. To get data from hive to a bag in pig
2. To store data from pig to a table in hive

```
grunt>A = load 'emp' using org.apache.hcatalog.pig.HCatLoader();
```

```
grunt>A = load 'dvs.emp' using org.apache.hcatalog.pig.HCatLoader();
```

```
grunt>B filter A by deptno==10;
```

```
grunt>store B into 'emp' using org.apache.hcatalog.pig.HCatStorer();
```

```
grunt>store B into 'dvs.emp' using org.apache.hcatalog.pig.HCatStorer();
```