********** TABLE TYPES **********

Managed or Internai 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(empnoint, ename string, job string, sal float, comm float, deptnoint) 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.

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_et(empnoint, ename string, job string, sal float, comm float, deptnoint) 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.

********** 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, sidint) partitioned by (yearint)row format delimited fields terminated by ',';

hive>load data local inpath '/home/training/dvs/student' into table std partition(year='2012');

hive>load data local inpath '/home/training/dvs/std_2011' into table st partition(year='2011');

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

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 countries data, static partitions will not work.

Dynamic Partitions:

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

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

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

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

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

Creating partition tables for multiple partitions:

hive>create table std(sname string, sidint) partitioned by (yearint, mop int, dopint) row format delimited feilds terminated by ',';

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

Adding partitions :

alter table std_dp add partition (year=2016);

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

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

Dropping partitions :

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

hive>alter table std_dp drop partition (year=2016,mop=02,dop=31);

Renaming Partitions :

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

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

hive>alter table std_dp exchange partition(year=2014) with table std_dp1;

Repairing or recovering partitions:

hive>alter table std_dp recover partitions;

or

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

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 year=2014">>sample.csv --appending the same file with more data

Writing HIVE script

cat>sample.hql

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

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

sethive.exec.dynamic.partition.mode=nonstrict;

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

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

Running hive script:

hive -f sample.hql

hive -f /home/training/hivescript.hql

Joins

1. EQUI or INNER Join

selectA.ename, B.dname, B.dloc from emp A join dept B ON (A.deptno=B.deptno);

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

2. Left outer Join

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

3. Right outer Join

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

4. Full outer Join

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

5. Cross Join

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

Views :

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';

Creatingview from a existing view :

create view emp_view1 as select * from emp_view;

Dropping a view

Drop view emp_view;

Passing parameters for single line commands thru Linux

tablename=emp;

col1=empno;

col2=ename;

no=4;

hive -e "select $col1, $col2 from $tablename limit $no";

Passing parameters for single line commands thru hive shell

set a=emp;

set b=dept;

set c1=empno;

select * from ${hiveconf:a};

Passing parameters for hql file

select ${hiveconf:col1}, ${hiveconf:col2} from ${hiveconf:tablename} limit ${hiveconf:limit_number};

hive -hiveconftablename=emp -hiveconf col1=empno -hiveconf col2=ename -hiveconflimit_number=4 -f h.hql;

hive -S -hiveconftablename=emp -hiveconf col1=empno -hiveconf col2=ename -hiveconflimit_number=4 -f h.hql;

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.csv' 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(empnoint, enamestring,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/hive/mapfile' 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(sidint, sname string, sub array<string>, pay_details map<string,int>, addrstruct<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], fee["fee"],fee["scholarship"], addr.city from struct_table;

Indexes :

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

COMPACT : This is Index type. It stores value & Block id .

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

BUCKETS

Bucketing is mainly for 2 reasons:

1. For faster joins(Join Optimization)

2. Table Sampling

Creating a bucketed table

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

Enabling Bucketing

hive>set hive.enforce.bucketing=true;

hive>insert into table emp_bk11 select * from emp;

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

Getting sample of 1 bucket data out of 3 buckets

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

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

Getting percentage wise sampling

hive>select * from emp_bktablesample(10 percent);

FILE FORMATS

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

TEXTFILE

RC

PARQUET

ORC

hive>create table emp(empnoint, ename string, job string, sal float, comm float, deptnoint) row format delimited fields terminated by ',' stored as textfiie;

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

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

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(empnoint, ename string, job string, sal float, comm float, deptnoint) row format delimited fields terminated by ',' stored as orcfile;

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

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

hive>create table emp_av(empnoint, ename string, job string, sal float, comm float, deptnoint) 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.

Miscellaneous:

Create a temporary table

hive>Create tempoarary table std_temp(sname string, sidint, yearint) 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(empnoint, ename string, job string, sal double, comm float, deptnoint) row format delimited fields terminated by ','
tblproperties('skip.header.line.count'='2','creator'='Rams','date'='2016-01-26');

Creating table with DB properties

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

Checking DB properties

hive>describe database extended dvs11;

Creating table with Table properties

hive>create table emp(ename string, empnoint) 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(empno as double) from emp;

Importing the structure of another table in hive

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

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 empl2 select * from empl 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 samelocation.

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 betweenPartition 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 wastetime. 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>create table emp_s(empnoint, ename string, job string, sal double, comm float, deptnoint, country string) skewed by (country) on ('US') stored as directories row format delimited fields terminated by ',';

hive>create table emp_s(empnoint, ename char(20), job char(20), sal double, comm float, deptnoint, 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

fromemp

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.

```
fromemp
insert overwrite directory '/dvs_hdfs/mngr_data' select * where job='MANAGER'
insert overwrite directory '/dvs_hdfs/clerk_data' select * where job='CLERK';
export HADOOP_HOME_WARN_SUPPRESS="TRUE"   --hadoop-env.sh
```