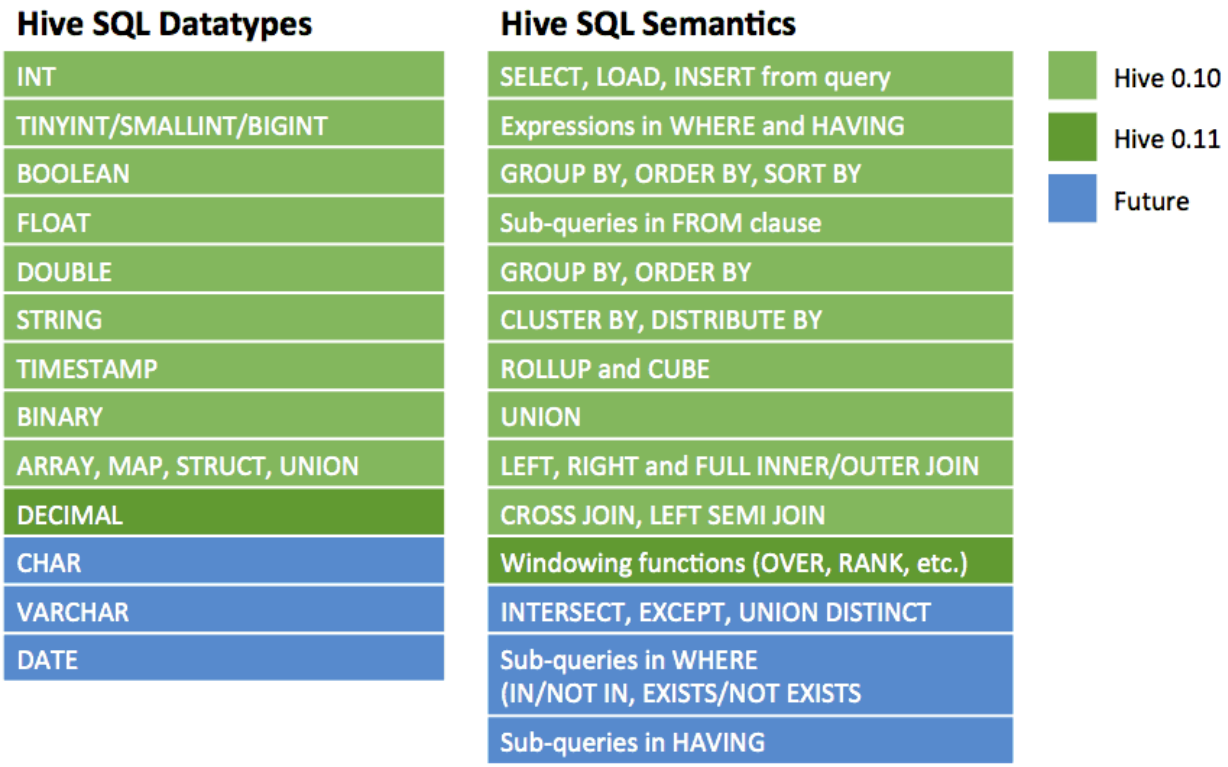
HIVE

HIVE FUNCTIONS



SHOW

```
hive> show databases;
hive> show tables;
hive> show databases like 'f*';
```

USE

```
hive> use default;
hive (default)> use test;
```

CREATE DATABASE

```
hive> create database if not exists financials;
hive> create database financials
comment 'holds financial tables'
```

```
location '/home/hduser/hive_db_dir';
```

PROPERTY TO PRINT CURRENT DATABASE

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

ALTER DATABASE

```
ALTER DATABASE financials SET DBPROPERTIES ('edited-by' =
'Jerry');
```

CREATE MANAGED TABLE

```
hive> create table records (id string, name string, att int,
marks int)
> row format delimited
> fields terminated by ',';

hive> create table employees (
> name string,
> salary float,
> subordinates array<string>,
> deductions map<string, float>,
> address struct<street:string, city:string, state:string,
zip:int>);

hive> create database financials2
> with dbproperties('creator' = 'Amit Mishra', 'date' = '2017-
05-19');

hive> create table records (year string, temperature int,
quantity int)
> row format delimited
> fields terminated by '\t';
```

```
hive> create table employees (
> name string,
> salary float,
> subordinates array<string>,
> deductions map<string, float>,
> address struct<street:string, city:string, state:string,
zip:int>);
```

EXTERNAL TABLE VS MANAGED TABLE

Hive has a relational database on the master node it uses to keep track of state. For instance, when you CREATE TABLE FOO(foo string) LOCATION 'hdfs://tmp/';, this table schema is stored in the database.

If you have a partitioned table, the partitions are stored in the database(this allows hive to use lists of partitions without going to the file-system and finding them, etc). These sorts of things are the 'metadata'.

When you drop an internal table, it drops the data, and it also drops the metadata.

When you drop an external table, it only drops the table with schema. It does not drops the metadata. That means hive is ignorant of that data now. It does not touch the data itself inside the metadata.

CREATE EXTERNAL TABLE

```
cat scripts/create_external_table.hql;
create external table if not exists mydb.stocks(
exchange string,
symbol string,
ymd string,
price_open float,
price_high float,
```

```
price_los float,
price_close float,
volume int,
price_adj_close float)
row format delimited fields terminated by ',' location
'/user/cloudera/myhive';
```

CREATING TABLE FROM EXISTING TABLE

```
hive> create external table stocks2
> like stocks;
```

CREATING EXTERNAL TABLES FROM MANAGED TABLES:

```
hive> create external table mydb.employees3
> like employees;
```

LOAD

```
hive> load data local inpath 'data/sample.txt'
> overwrite into table records;
hive> load data local inpath 'data/NYSE_daily'
> overwrite into table stocks;
```

COPY DATA FROM ONE TABLE TO ANOHTER

```
hive> insert overwrite table nyse_daily_bak select * from
nyse_daily;
NOTE: Launges Map-only job
hive> insert overwrite dividends_bak select * from dividends;
```

DROP

```
hive> drop table records;
hive> drop database if exists financials2 cascade;
```

--NOTE: cascade/restrict is optional

QUIT

```
hive> quit;
```

SELECT

```
hive> select year, MAX(temperature) from records
> group by year;
```

DESCRIBE

```
hive> describe database financials;
hive> describe database default;
hive> describe database mydb;
hive> describe employees;
hive> describe extended employees;
hive> describe formatted employees;
```

DESCRIBE SPECIFIC FIELD

```
hive> describe employees.address;
```

DESCRIBE EXTENDED

```
hive> describe database extended financials2;
```

ALTER

```
hive (financials2)> alter database financials set
dbproperties('modified by' = 'AMIT MISHRA');
```

CLONE SCHEMA (DATA IS NOT COPIED)

```
hive> create table employeedb.employees1
> like employees;
```

CLONE SCHEMA TO ANOTHER DB

```
hive> create table financials.employees3 like employees;
hive> load data local inpath
'/home/hduser/hivedata/data/NYSE_daily' into table stocks2;
hive> drop database mydb;
FAILED: InvalidOperationException(message:Database mydb is not
empty)
hive> drop database mydb cascade;
```

$ cat scripts/stocks.hql;

```
hive> create external table if not exists mydb.stocks(exchange
string,
symbol string,
ymd string,
price_open float,
price_high float,
price_los float,
price_close float,
volume int,
price_adj_close float)
row format delimited fields terminated by '\t' location
'/user/cloudera/myhive';
```

```
$ cat scripts/load_stocks.hql;
hive> load data local inpath 'data/NYSE_daily'
overwrite into table mydb.stocks;
```

Hive Regular Expression

ARITMETIC OPERATORS:

+, -, *, /, %, &(AND), |(OR), ^(XOR),~(NOT)

```
hive> create table mydb.employees2 like employees;

hive> select symbol, `price.*` from stocks limit 5;

hive> select upper(name),salary,deductions["Federal
Taxes"],round(salary*(1-deductions["Federal Taxes"]))

hive> select count(*) as count, avg(salary) as salary from
employees_no_partition;
```

AGGREGATE FUNCTIONS

count, sum, avg, min, max ...

```
hive> select count(*), avg(salary) from employees_no_partition;

hive> set hive.map.aggr = true;

hive> select MAX(column_name) from tablename;

hive> select MIN(column_name) from tablename;
```

CASE..WHEN..THEN

```
hive> select emp_name, salary,

case

when salary <=100000 then 'low'

when salary >=300000 and salary <400000 then 'middle'

when salary >=400000 and salary <=500000  then 'high'

else 'very high'

end

as salarybracket
```

```
from employee;
```

HIVE WORDCOUNT

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS

SELECT word, count(1) AS count FROM

(SELECT explode(split(line, '\s')) AS word FROM docs) w

GROUP BY word

ORDER BY word;
```

PARTITIONING IN HIVE

Table partitioning means dividing table data into some parts based on the values of particular columns like date or country, segregate the input records into different files/directories based on date or country.

Partitioning can be done based on more than column which will impose multi-dimensional structure on directory storage. For Example, In addition to partitioning log records by date column, we can also sup divide the single day records into country wise separate files by including country column into partitioning. We will see more about this in the examples.

Partitions are defined at the time of table creation using the **PARTITIONED BY** clause, with a list of column definitions for partitioning.

SYNTAX

```
CREATE [EXTERNAL] TABLE table_name (col_name_1 data_type_1,
....)

PARTITIONED BY (col_name_n data_type_n [COMMENT col_comment],
...);
```

ADVANTAGES

Partitioning is used for distributing execution load horizontally.

As the data is stored as slices/parts, query response time is faster to process the small part of the data instead of looking for a search in the entire data set.

For example, In a large user table where the table is partitioned by country, then selecting users of country ‘IN’ will just scan one directory ‘country=IN’ instead of all the directories.

LIMITATIONS

Having too many partitions in table creates large number of files and directories in HDFS, which is an overhead to NameNode since it must keep all metadata for the file system in memory only.

Partitions may optimize some queries based on Where clauses, but may be less responsive for other important queries on grouping clauses.

In Mapreduce processing, Huge number of partitions will lead to huge no of tasks (which will run in separate JVM) in each mapreduce job, thus creates lot of overhead in maintaining JVM start up and tear down. For small files, a separate task will be used for each file. In worst scenarios, the overhead of JVM start up and tear down can exceed the actual processing time.

MANAGED PARTITIONED TABLE

```
CREATE TABLE partitioned_table(
     firstname STRING,
     lastname  STRING,
     address   STRING,
     city        VARCHAR(64),
     email     STRING,
     web       STRING
     )
     PARTITIONED BY (country VARCHAR(64), state VARCHAR(64))
     ROW FORMAT DELIMITED
     FIELDS TERMINATED BY ‘,’
     LINES TERMINATED BY ‘\n’
     STORED AS SEQUENCEFILE;
hive> DESCRIBE FORMATTED partitioned_table;
```

Partitioned columns country and state can be used in Query statements **WHERE** clause and can be treated regular column names even though there is actual column inside the input file data.

External Partitioned Tables

We can create external partitioned tables as well, just by using the **EXTERNAL** keyword in the CREATE statement, but for creation of External Partitioned Tables, we do not need to mention

LOCATION clause as we will mention locations of each partitions separately while inserting data into table.

STATIC PARTITIONING

In this mode, input data should contain the columns listed only in table definition (for example, firstname, lastname, address, city, post, phone1, phone2, email and web) but not the columns defined in partitioned by clause (country and state).

If our input column layout is according to the expected layout and we already have separate input files for each partitioned key value pairs, like one separate file for each combination of country and state values (country=IN and state=DL), then these files can be easily loaded into partitioned tables with below syntax.

LOADING DATA

```
hive> LOAD DATA LOCAL INPATH
'hdfs://localhost:9000/hadoop/hive/part.csv'
      INTO TABLE partitioned_table
      PARTITION (country = 'IN', state = 'DL');
```

LOADING PARTITION FROM OTHER TABLE

We can load or add partitions with query results from another table as shown below.

```
hive> INSERT OVERWRITE TABLE partitioned_table
      PARTITION (country)
      SELECT * FROM pseudo_table test
      WHERE test.country = 'IN' AND test.state = 'DL';
```

AIRLINE DATA PARTITIONING

```
create schema airline;
use airline;
```

// USE DATASET DELAYEDFLIGHTS.CSV

hive> create table airline.ontimeperfect

(DayofMonth INT ,

DayOfWeek INT ,

DepTime INT ,

CRSDepTime INT ,

ArrTime INT ,

CRSArrTime INT ,

UniqueCarrier STRING ,

FlightNum INT ,

TailNum STRING ,

ActualElapsedTime INT ,

CRSElapsedTime INT ,

AirTime STRING ,

ArrDelay INT ,

DepDelay INT ,

Origin STRING ,

Dest STRING ,

Distance INT ,

TaxiIn STRING ,

TaxiOut STRING ,

Cancelled INT ,

CancellationCode STRING ,

Diverted INT ,

CarrierDelay STRING ,

WeatherDelay STRING ,

NASDelay STRING ,

SecurityDelay STRING ,

```
LateAircraftDelay STRING)
PARTITIONED BY (Year INT, Month INT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

To load partitioned table we use below command:

```
hive> SET hive.exec.dynamic.partition = true;
hive> SET hive.exec.dynamic.partition.mode = nonstrict;
hive> INSERT OVERWRITE TABLE airline.ontimeperfection
PARTITION(Year, Month) SELECT DayofMonth, DayOfWeek, DepTime,
CRSDepTime, ArrTime, CRSArrTime, UniqueCarrier, FlightNum,
TailNum, ActualElapsedTime, CRSElapsedTime, AirTime, ArrDelay,
DepDelay, Origin, Dest, Distance, TaxiIn, TaxiOut, Cancelled,
CancellationCode, Diverted, CarrierDelay, WeatherDelay,
NASDelay, SecurityDelay, LateAircraftDelay, Year, Month FROM
stg_airline.onTimePerf;
```

BUCKETING IN HIVE

Partitioning gives effective results when,

- There are limited number of partitions
- Comparatively equal sized partitions

But this may not possible in all scenarios, INSTANCE, when are partitioning our tables based locations like country, some countries if comprise large partitions (ex: 4-5 countries itself contributing 70-80% of total data) where as small countries data will create small partitions (remaining all countries in the world may contribute to just 20-30 % of total data).

So, In these cases **Partitioning will not be ideal.**

HENCE WE REQUIRE BUCKETING

FEATURES

- Bucketing concept is based on (hashing function on the bucketed column) mod (by total number of buckets). The hash_function depends on the type of the bucketing column.

- Records with the same bucketed column will always be stored in the same bucket.
- We use **CLUSTERED BY** clause to divide the table into buckets.
- Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based.
- Bucketing can be done along with Partitioning on Hive tables and even without partitioning.
- Bucketed tables will create almost equally distributed data file parts.

ADVANTAGES

- Bucketed tables offer efficient sampling than by non-bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge.
- As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non-bucketed tables. In Map-side join, a mapper processing a bucket of the left table knows that the matching rows in the right table will be in its corresponding bucket, so it only retrieves that bucket (which is a small fraction of all the data stored in the right table).
- Similar to partitioning, bucketed tables provide faster query responses than non-bucketed tables.
- Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map-side joins even more efficient, since the join of each bucket becomes an efficient merge-sort.

CREATION OF BUCKETED TABLES

We can create bucketed tables with the help of CLUSTERED BY clause and optional SORTED BY clause in CREATE TABLE statement. With the help of the below HiveQL we can create bucketed_user table with above given requirement.

```
CREATE TABLE bucketed_table(
        firstname VARCHAR(64),
```

```
  lastname  VARCHAR(64),
  address   STRING,
  city        VARCHAR(64),
  state     VARCHAR(64),
  email     STRING,
  )
COMMENT 'A bucketed sorted user table'
PARTITIONED BY (country VARCHAR(64))
CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS
STORED AS TEXTFILE;
```

Unlike partitioned columns (which are not included in table columns definition), Bucketed columns are included in table definition as shown in above code for state and city columns.

Inserting data Into Bucketed Tables

Similar to partitioned tables, we can not directly load bucketed tables with LOAD DATA (LOCAL) INPATH command, rather we need to use INSERT OVERWRITE TABLE … SELECT …FROM clause from another table to populate the bucketed tables. For this, we will create one temporary table in hive with all the columns in input file from that table we will copy into our target bucketed table.

Lets assume we have created temp_user temporary table, and below is the HiveQL for populating bucketed table with temp_user table.

To populate the bucketed table, we need to set the property hive.enforce.bucketing = true, so that Hive knows to create the number of buckets declared in the table definition.

```
set hive.enforce.bucketing = true;
```

LOADING DATA

```
hive> LOAD DATA LOCAL INPATH
    > 'hdfs://localhost:9000/hadoop/hive/part.csv'
    > INTO TABLE partitioned_table
```

```
    > PARTITION (country = 'IN', state = 'DL');
```

DATA INSERT FROM ONE TABLE TO BUCKET_TABLE

```
hive> INSERT OVERWRITE TABLE bucketed_user PARTITION (country)
 > SELECT firstname, lastname, address, city, state, post,
 > phone, email, web, country, FROM temp_table;
```

set hive.enforce.bucketing = true;

The property hive.enforce.bucketing = true similar to hive.exec.dynamic.partition=true property in partitioning. By Setting this property we will enable dynamic bucketing while loading data into hive table.

It will automatically sets the number of reduce tasks to be equal to the number of buckets mentioned in the table definition (for example 32 in our case) and automatically selects the clustered by column from table definition.

If we do not set this property in Hive Session, we have to manually convey same information to Hive that, number of reduce tasks to be run (for example in our case, by using set mapred.reduce.tasks=32) and CLUSTER BY (state) and SORT BY (city) clause in the above INSERT …SELECT statement at the end.

```
hive> SELECT firstname, country, state, city FROM bucketed_user
    > TABLESAMPLE(BUCKET 4 OUT OF 4 ON state);
hive> SELECT firstname, country, state, city FROM temp_user
LIMIT 129 ;
hive> SELECT firstname, country, state, city FROM bucketed_user
TABLESAMPLE(1 PERCENT);
```

HIVE TO STOP MAP REDUCE

```
hive> set hive.exec.mode.local.auto = true;
```

BUKETING CLI COMMANDS TO REMOVE WARN & ERROR

```
set hive.exec.dynamic.partition=true;

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

set hive.exec.max.dynamic.partitions=1000;

set hive.exec.max.dynamic.partitions.pernode=1000;

set hive.enforce.bucketing = true;
```

STEPS:

---

1. sudo nano bucket.csv

2. hadoop fs -copyFromLocal /home/hduser/bucket.csv /hadoop/hive/

3. hive

On HIVE prompt:-

```
hive>create table emp_bucket (id int,first_name STRING,last_name STRING,email STRING,city STRING,country STRING) partitioned by (state STRING) clustered by (id) into 4 bucket row format delimited fields terminated by ',';

hive> desc emp_bucket;
```

O/P

```
OK
id                      int
first_name              string
last_name               string
email                   string
city                    string
country                 string
state                   string

# Partition Information
```

```
# col_name                data_type                comment

state                     string
```

```
hive>  LOAD DATA INPATH
'hdfs://localhost:9000/hadoop/hive/bucket.csv' INTO TABLE
emp_bucket parttion (state = 'delhi');
```

Location of DB & Table in Hive

```
3. hadoop fs -ls
hdfs://localhost:9000/user/hive/warehouse/hive_db1.db/emp_bucket
```

```
4. hadoop fs -ls
hdfs://localhost:9000/user/hive/warehouse/hive_db1.db/emp_bucket
/state=delhi
```

```
5. hadoop fs -cat
hdfs://localhost:9000/user/hive/warehouse/hive_db1.db/emp_bucket
/state=delhi/bucket.csv
```

RUN HIVE IN CLI

```
usage: hive
 -d,--define <key=value>          Variable substitution to apply to Hive
                                  commands. e.g. -d A=B or --define A=B
 -e <quoted-query-string>         SQL from command line
 -f <filename>                    SQL from files
 -H,--help                        Print help information
 -h <hostname>                    Connecting to Hive Server on remote host
    --hiveconf <property=value>   Use value for given property
    --hivevar <key=value>         Variable substitution to apply to hive
                                  commands. e.g. --hivevar A=B
 -i <filename>                    Initialization SQL file
 -p <port>                        Connecting to Hive Server on port number
 -S,--silent                      Silent mode in interactive shell
 -v,--verbose                     Verbose mode (echo executed SQL to the
                                  console)
```

Create a FILE

TYPE all the Hive Query Language (which is required for process program)

Run HIVE query from CLI:

```
$ hive –e ‘hive commands’
$ hive -e 'select a.col from tab1 a'
```

Example of dumping data out from a query into a file using silent mode

```
$ hive -S -e 'select a.col from tab1 a' > a.txt
```

Example of running a script non-interactively from local disk

```
$ hive -f /home/my/hive-script.sql
```

Example of running a script non-interactively from a Hadoop supported filesystem (starting in Hive 0.14)

```
$ hive -f hdfs://<namenode>:<port>/hive-script.sql/.hql
$ hive -f s3://mys3bucket/s3-script.sql/.hql
```

Example of running an initialization script before entering interactive mode

```
$ hive -i /home/my/hive-init.sql
```