Hive

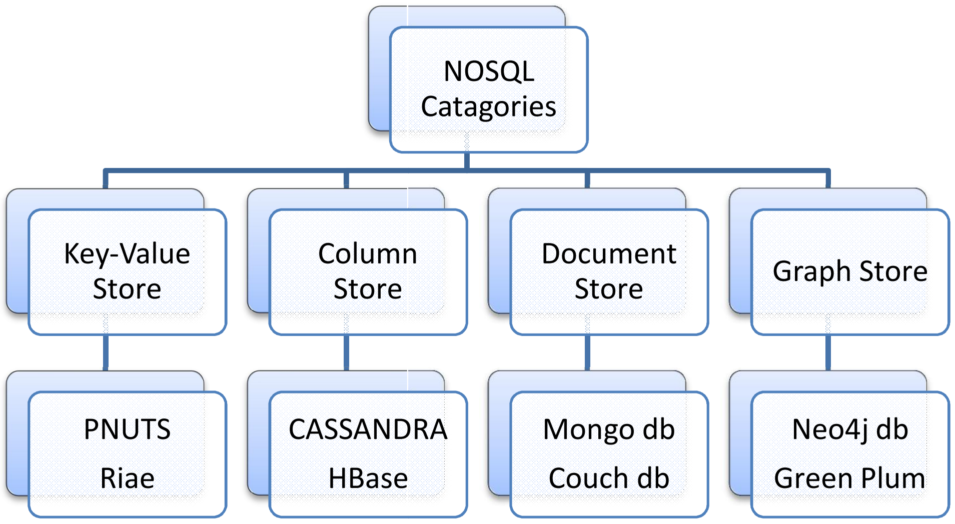
Hive is Data-Warehousing environment in hadoop s/w.
Hive is used to process and store structured, unstructured, semi structured data formats.
Hive has built in functions to process data and UDF'S to develop custom functions.
Hive has HQL (Hive Query language) to process data.
Differences between HQL And SQL….

hql	sql
DDL	DDL
DML	DML
-	DCL
-	TCL
Row level insertions are not possible(Bulk	Row level insertions are possible
Updates and Delete Commands are not possible	Update and Delete Commands are available

Hive can import data from local files as well as HDFS files. The back store of hive table is HDFS files. i.e Hive will run on top of HDFS.
Hive can import data from RDMS tables using SQOOP.
(SQOOP = SQL + HADOOP)
When you submit the hql query hadoop stream will convert the query into java MR code and map reduce jobs will be submitted.

Hive → Hive compiler →Hadoop Stream→Java MR code→Jobtracker→job Execution

We can store HQL results directly into the HDFS file and local file.
We can export the results of hql into RDBMS table using SQOOP.
Hive basically doesn't have row level operations such as inserts, updates and deletes so hive doesn't contain complete features of RDBMS. But in hadoop, NOSQL db like Cassandra, Hbase which comes under columnar – store category of NoSql.
Hbase has complete row level control such as row level insertions, deletions and updates.
By integrating Hive with Hbase we can extend the functionality. By using this facility hql can process hbase data without writing complex java programming.
From hive 0.7 onwwards hive is able to integrate with Non Hadoop -Nosql databases such as PNUTS, GreenPlum, Mongodb, Couch db ete.
Hive doesn't support OLTP operations but Nosql will support.
The NoSql's data can be processed by Hive once it is integrated.



Basic Commands of Hive
Hive> show tables;
Hive> create database **mydb**;
Hive>use database mydb;
Hive> create database **hivedb**;
Hive>use database hivedb;
Hive> show databases;
    Mydb;
    Hivedb;
Hive> create table sample(a int, b int, c int);
Hive> create table hivedb.trans(a int, b int);
Hive> create table mydb.trans(a int, b int);
Hive> create table mydb.sample1(x int);
Hive> drop table mydb.trans; (first delete the table then delete the db)
Hive> drop database mydb; (deleting database);
Hive> create table emp(eno int, ename string);
Ok;
(The row level insertions are not possible here in hive, so we need to load the table from files)
Hive> quit;

Creating sample file:
$ gedit file1
$ cat file1
Note: 1001 octal code for delimiter.
<u>Creating Hive tables….</u>
Hive> create table table1(f1 datatype, f2 <datatype>, ……);
<u>Loading data from local file to hive table….</u>
Hive> load data local inpath '<file path>' into table <hive table>;
Hive> load data local inpath 'f2' into table table1;
Hive> load data local inpath 'fs' overwrite into table table1;
<u>Loading data from HDFS file into hive table….</u>
Hive> load data inpath 'myfile' into table table1;
Hive> load data inpath 'myfile1' overwrite into table table1;
Note:- Once hdfs file is copied into the table, the hdfs file will be erased.
Files

```
1011120000
1021230000
1031340000
```

Hive>create table tmp(str string);
Hive> load data local inpath 'file1' into table tmp;
Hive> create table tmp emp(ecode string, dno string, esal int);
<u>Table to table copy</u>
Insert overwrite into table emp
Select Substr(str, 1, 3), Substr(str, 4, 2), Substr(str, 6, 5) from tmp;
Hive> select * from emp;

101	11	20000
102	12	30000

File1

```
101,Arun,20000,11,m
102, Amar, 30000,12,f
```

```
Hive> CREATE TABLE staff(ecode STRING, ename STRING, esal, STRING, dno STRING, sex STRING)
          ROW FORMAT DELIMITED
          FIELDS TERMINATED BY ‘,’;
Hive> load data local inpath ‘file1’ into table staff;
```

Web log Analytics(Web Mining);

http://ABC:/page1/a=100&b=200&c=300

```
localhost~]$hive –e showtables;
```

Data Types In Hive:

32 bit –int
64 bit- long
32 bit decimal-float
64 bit decimal-double
Alphanumeric-STRING
True/false- BOOLEAN

Complex data types in PIG:

Struct
Map
Array

Creating table with array column:

```
Hive> create table sample(price array<int>);
Hive>
CREATE TABLE emp(ecode STRING, ename STRING,
                        ph ARRAY <STRING>,
                        query ARRAY<STRING>,
                        sex STRING)
hive>
CREATE TABLE url(host STRING, path STRING,
                   query MAP<STRING, STRING>);
```

MAP<data type, data type>
MAP<key, value>

URL log:-

http://yahoo.com/p1/? user=user& pwd=abc & loc=hyd

http://google.com/p3/? user= uvwe & pwd = abc &loc = pune

hive> CREATE TABLE urltab (url STRING);
hive> LOAD DATA LOCAL INPATH 'urllog' INTO TABLE urltab;
hive> CREATE TABLE hpq(host STRING, path STRING, query STRING);
hive> INSERT OVERWRITE TABLE hpq
SELECT parse_url(url, 'HOST'), parse_url(url, 'PATH', parse_url(url, 'QUERY');
Hive> SELECT * FROM hpq;

host	path	Query
Yahoo.com	P1	user=user& pwd=abc & loc=hyd
Google.com	P2	user= uvwe & pwd = abc &loc = pune

Hive> CREATE TABLE hpqmap(host STRING, path STRING, gmap MAP<STRING, STRING>);
Hive> INSERT OVERWRITE TABLE hpqmap
SELECT host, path, Str_to_map(query, 'q') from hpq;
Hive> SELECT * FROM hpqmap;

host	path	Query
Yahoo.com	P1	[user="user", pwd="abc", loc="hyd"]
Google.com	P2	[user=" uvwe" , pwd = "abc", loc = "pune"]

Hive> CREATE TABLE TARGET (host STRING, path STRING, user STRING, loc STRING);
HIVE>INSERT OVERWRITE TABLE TARGET
SELECT host, path, gmap[user], gmap[loc] from hqmap;
Hive> SELECT * FROM target;
STRUCTURE
HIVE> CREATE TABLE sample2(a INT, b INT, c STRUCTURE(x STRING, y INT, z INT), d INT);
HIVE> SELECT a, b, c[x], c[y], c[z], d;

MAP
ARRAY(array is simple collection)
STRUCT(1)

NAME	SEX	DNO
A	M	12
B	F	11
c	M	18
d	M	11
e	F	32

HIVE> CREATE TABLE emp2 LIKE emp;

Joinging two tables placed in other tables

Hive> INSERT OVERWRITE TABLE emp2
SELECT * FROM (SELECT name, 'FEMALE', dno from emp WHERE sex='F'

```
 UNION ALL
 SELECT name, ‘MALE’, dno from emp WHERE sex=’M’ ) staff;
```

```
Hive>INSERT OVERWRITE TABLE emp2
        SELECT * FROM(SELECT name, ‘MALE’,’MARKETING’ FROM emp WHERE sex= ‘M’ and dno=’11’
        UNION ALL
        SELECT name, ‘MALE’, ‘HR’ FROM emp where sex=’M’ and dno=12
        UNION ALL
        SELECT name, ‘FEMALE’, ‘HR’ FROM emp WHERE sex= ‘F’ and dno=18)
        Staff;
Hive>use dbname;
Hive>show tables;
```

In the file if the quality is 0,1,3 is good otherwise bad(quality indicator)

```
ABCDEF1965DEF25SDSE0DSDS
BSDASD1954FDS45VDFF3SSFSF
FDFGFFS1986DF34GDSD4GDFS
```

```
Hive>CREATE TABLE temp(y STRING, t INT, q INT);
HIVE>INSERT OVERWRITE TABLE Temperature
        SELECT * FROM (SELECT y, t, ‘good’ FROM temperature WHERE Q in(0,1,3)
        UNION ALL
        SELECT y,t, ‘bad’ FROM temperature WHERE q not in (0,1,3) tx;
```

Aggregate functions in HIVE : COUNT(*), SUM(), AVG(), MAX(), MIN()
Statistical Aggragations: **STDDEV, COVARIANCE, VARIANCE**

```
Hive>SELECT COUNT(*) FROM emp;
Hive>SELECT SUM(esal), AVG(esal), MAX(esal), MIN(esal) FROM emp  WHERE city =’hyd’;
Hive>SELECT STDDEV(IBMstock), STDDEV(intostock) FROM stock;
```

ORDER BY clause: used to sort the rows of table

```
Hive> SELECT * FROM emp ORDER BY ename;
Hive> SELECT * FROM emp ORDER BY esal DESC;
HIVE> SELECT * FROM emp ORDER BY esal DESC, dno, sec DESC;
```

LIMIT clause:

```
Hive> SELECT * FROM tmp LIMIT 10;
Hive>SELECT * FROM emp ORDER BY esal DESC LIMIT 10;
```

DISINCT() : eliminates duplicates from the list not from the table.

```
Hive> SELECT DISTINCT(dno) FROM emp;
```

To ensure duplicates existing in a column

```
Hive> SELECT COUNT(*) DISTINCT(dno) FROM  emp;
```

If result is more than zero, yes duplicates are available.
If duplicates are there, what are the duplicates?

GROUP BY clause:

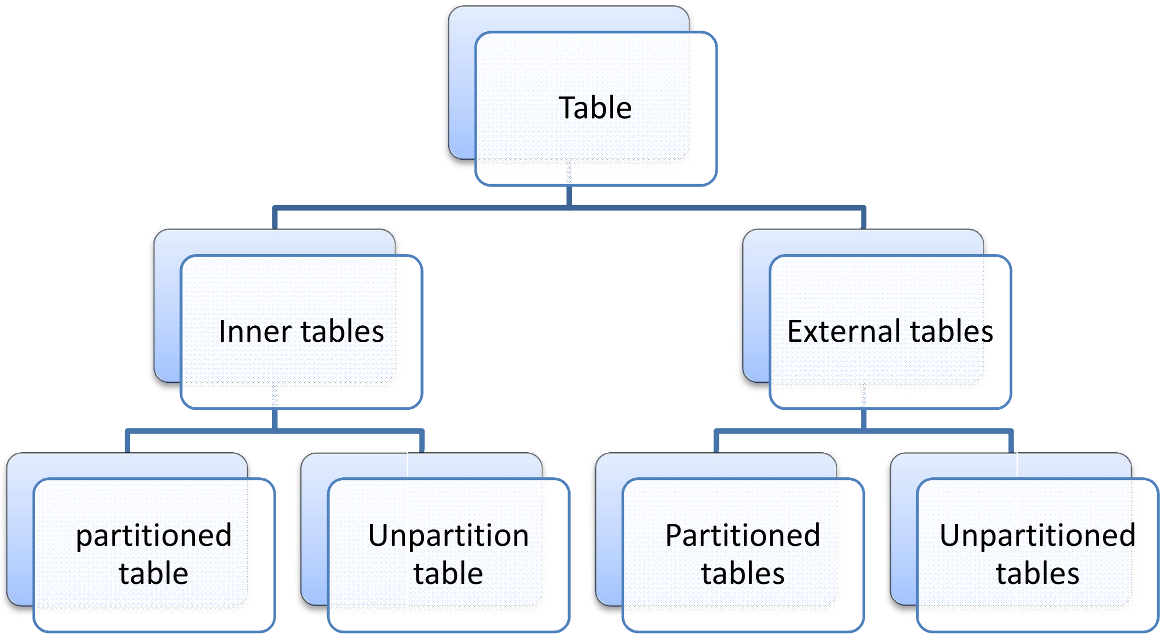
```
Hive>SELECT ecode, COUNT(*) FROM tmp GROUP BY HAVING COUNT(*)>1;
```

Hive>SELECT dno, sex, SUM(esal) FROM emp GROUP BY dno, sex HAVING dno in('11','13');

Difference between WHERE clause and HAVING clause ?
HAVING clause is more faster than WHERE clause. For GROUPING HAVING Is more preferable.

LIKE operator:
Hive> SELECT * FROM tmp WHERE ename LIKE 'K%';
Hive> SELECT * FROM tmp WHERE ename NOT LIKE 'K%';
Hive> SELECT * FROM tmp WHERE ename LIKE '%K';
Hive> SELECT * FROM tmp WHERE ename LIKE '%K%';
Hive> SELECT * FROM tmp WHERE ename LIKE 'K_ _ _';
Hive> SELECT * FROM tmp WHERE ename LIKE '_ _ K_';
Hive> SELECT * FROM tmp WHERE ename LIKE '_K%';
Hive> SELECT * FROM tmp WHERE ename LIKE '%K_';

TABLE types



Hadoop fs -ls /user/hive/warehouse/ptab;
Partitioned and Unpartitioned tables:
By default each table is non partitioned tables Hive creates a single file for entire data of a table.
Partitioned table : hive creates separate files for each partition
Creating Partitioned table:
Hive> CREATE TABLE ptab(str STRING) PARTITIONED BY(ds STRING);
Str- column of the string
Ds-partitioned column which is logical but we can use this column in your select query)

Creating Multiple Partitioned table:
Hive>CREATE TABLE xtab(str STRING) PARTITIONED BY (d STRING, m STRING, y STRING);
Hive>LOAD DATA LOCAL INPATH 'filex' INTO TABLE xtab PARTITION(d='1', m= '1', y= '12');

```
HIVE> SELECT * FROM xtab WHERE (y='2003' and m>3)
Hive>SELECT * FROM xtab WHERE  y in('2004', '2008', '2006', '2007','2005');
HIVE> SELECT * FROM xtab WHERE (y='2003' and m<'7');
```

<u>Non Partitoned Table to Partitioned Table Copy:</u>

<u>Step1:</u>

```
Hive> CREATE TABLE emp(ecode STRING, ename STRING, esal INT, dno STRING, sex STRING);
        ROW FORMAT DELIMITED BY
        FIELDS TERMINATED BY ',';
```

<u>Step2:</u>

```
Hive>LOAD DATA INPATH ' staff.txt' INTO TABLE emp;
```

<u>Step3:</u>

```
Hive>CREATE TABLE staff(ecode STRING, ename STRING, esal INT, dno STRING, sex STRING)
       PARTITIONED BY (dept STRING);
```

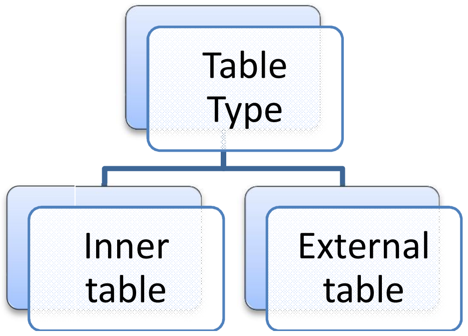
<u>Step4:</u>

```
Hive>INSERT OVERWRITE TABLE staff PARTITION(dept='11')
       SELECT * FROM emp WHERE dno='11';
Hive>INSERT OVERWRITE TABLE staff PARTITION(dept='12')
       SELECT * FROM emp WHERE dno='12';
```

<u>Partitoned Table to Partitioned Table Copy:</u>

Table1→partitioned by dept
Table2→partitioned by gender

```
Hive>INSERT OVERWRITE table2 PARTITION (gender='f')
       SELECT * FROM table1 where sex='f'; (Reading all partitions loading into Gender='f' partition)
Hive>INSERT OVERWRITE TABLE table2 PARTITION (gender='m')
       SELECT * FROM table1  WHERE sex='m'; (Reading all partitions loading into gender='m' partition).
Hive>INSERT OVERWRITE TABLE tab2 PARTITION (gender='m', dept='c2')
        SELECT * FROM table1 WHERE dept ='c2' and sex='f';
```



Location of Inner table- /user/hive/warehouse/table()directory.

Location of External table - We choose the location of working 'mydata'- user/training/mydata/tab.

CREATE TABLE sample(x STRING);   CREATE EXTERNAL TABLE mytable(x INT, y INT) LOCATION 'my data';

Dropping Inner table:

Hive> drop table sample; if table is dropped then automatically the backend data is also deleted. (MetaData and Physical data is deleted). WareHouse location. Nonpartitioned inner table.

Dropping External table:

Hive>drop table mytable; if table is droped, back end data is safe(MetaData only deleted but physical data is safe). Custom Location. Partitioned External table.

Joins:

To combine Columns of two or more tables

To collect information from more than one table

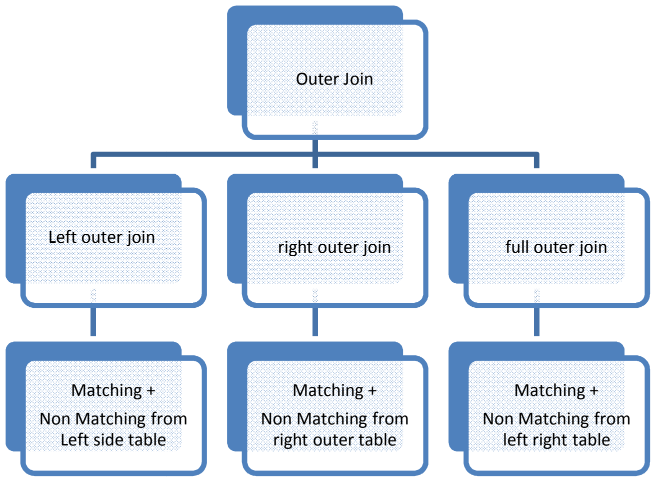
Ecode	Ename	Esal	Dno
101	Prasad	30000	11
102	Banu	32000	12
103	Karthik	43000	12
104	Chandu	32000	11
105	Kiran	25000	17
106	Sanju	32000	18

Dno	Dname	Dlocation
11	Marketing	Hyd
12	HR	Delhi
13	Finance	Hyd
20	Production	Delhi
21	Admin	Hyd

Joins:

Inner Joins→Only marketing rows with join condition will be merged

Outer Join→Matching and non matching rows with condition will be merged.


Outer Join
Left outer join
right outer join
full outer join
Matching + Non Matching from Left side table
Matching + Non Matching from right outer table
Matching + Non Matching from left right table

Join on Emp.dno=Dept.dno

Inner Join:

SELECT e.ecode, e.ename, e.esal, d.dname, d.loc FROM emp e JOIN dept d ON(e.dno=d.dno);

SELECT e.ecode, e.ename, e.esal, d.dname, d.loc FROM emp e LEFT OUTER JOIN dept d ON(e.dno=d.dno);

SELECT e.ecode, e.ename, e.esal, d.dname, d.loc FROM emp e RIGHT OUTER JOIN dept d ON(e.dno=d.dno);

SELECT e.ecode, e.ename, e.esal, d.dname, d.loc FROM emp e FULL OUTER JOIN dept d ON(e.dno=d.dno);

Creating views for Joins:

```
Hive> CREATE VIEW empinfo AS

        SELECT e.ecode, e.ename, e.esal, d.dname, d.dloc

        FROM emp e FULL OUTER JOIN dept d ON (e.dno=d.dno);

Hive> INSERT OVERWRITE TABLE empinfo

        SELECT e.ecode, e.ename, e.esal, d.dname, d.dloc

        FROM emp e FULL OUTER JOIN dept d ON (e.dno=d.dno);
```

UDF (user defined functions)

Map Reduce Job API-$ ls /usr/lib/hadoop-0.20/hadoop-core.jar

Jar for hive api- $ ls /usr/lib/hive/lib/hive-exe.0.7.1-cdh302.jar

Jar for pig api- $ /usr/lib/pig/pig-core.jar

Registering Jar file in Hive:

Hive> add jar /home/training/desktop/HiveUdf.jar;

Creating temporary file to hive for resgitered jar

Hive> create temperory function todate as ‘udf.hive.functions.UdfClass’;

Created function name=todates and class name in UDF file = udf.hive.functions.UdfClass;

Exampleof calling the function (UDF)

Hive> select v, todate(v) from tx;

Registering Jar in Pig:

```
$ pig

Grunt> register  /home/training/desktop/pigUDF.jar;

Grunt> define def1 a.b.c.pigfunc();
```

Calling the Function:

```
Grunt> y = FOREACH x GENERATE str, def1(str) AS str2;

Grunt> z = FOREACH x GENERATE str, a.b.c.pigfunc(str) AS  str2;
```

Save this file as xmltmp   (str STRING)

```
<r><a id = ‘101’> 100 </a> <b  id = ‘102’> 200 </b> </r>

<r><a id = ‘102’> 250 </a> <b  id = ‘103’> 350 </b> </r>
```

It’s not a meaningful data.

```
Hive>SELECT xpath(str,  ‘r/a/text()’) FROM xmltmp;

[‘100’, ‘200’]

[‘250’, ‘350’]

Hive>SELECT xpath(str, ‘r/a/@id’) FROM xmltmp;

[‘101’, ‘102’]

[‘102’, ‘103’]

Hive> SELECT xpath(str, ‘r/*[@id = “102”]/text()’) FROM xmltmp;

[‘200’]

[NULL]

Hive> SELECT xpath_boolean(str, r/*[@id = “102”]) FROM xmltmp;

[TRUE]

[FALSE]

Hive> SELECT count(*) FROM xmltmp WHERE xpath_boolean(str, ‘r/*[@id=’102’]) = TRUE;
```

Xpath →returns string array

Xpath_int→returns integer value

Xpath_float→returns float value

Xpath_double→returns double value

Xpath_boolean→returns Boolean values

Xpath_long→returns long values

explode : it converts elements of an array into multiple rows. Ie it is again generating a table.

This functions comes under category called UDTF(User defined Table-generated functions)

Str[“10”,”20’,”30”]

SELECT explode(str) AS val FROM tab;   // val is a column alias which is mandatory.

10

20

30

UDTF: it had to be applied with lateral view, which is used to construct temporary views. Views will be expired after completion of query execution.

A	B
1	[“100”, “200”, “300”]
2	[“300”, “400”]
3	[“500”]

<u>LATERAL VIEW:</u>

1	100
1	200
1	300
2	300
2	400
3	500

Size() : counts number of elements in an array or a map collection.

Example: str[“a”,”b”, “c”]

Hive>select size(str) from tab;

Array elements can be accessed using index numbers.

Index number starts with 0 and last elemnt indexis n-1.

Hive> SELECT xpath(str, ‘r/*/text()’)[0], xpath(str, ‘r/*/text())[1] FROM tab;

UDTF:

explode

json_tuple

parse_url_tuple

JSON loading and parsing in hive:

Simple json record looks like this

```
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 25,
    "height_cm": 167.64,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax",  "number": "646 555-4567" }
    ]
}
```

{ record1
“field1” : “value1”;
“field2” : ”value2”;
“field3” : “ value3”;
}
{ record2
“field1” : “v1”;
“field2” : ”v2”;
“field3” : “ v3”;
}

Field1	Field2	Field3	Field4
Value1	Vlaue2	Value3	Value4
V1	V2	V3	V4

Latest weblogs are maintained in json formats.
Sample weblog in text format:
192.300.100.73**google.com****mypage2013-1-2
JSONfile : sample.json
```
{"a":200, "b" : 300, "c" : 400}
{"a":300, "b" : 350, "c" : 250}
```
1st model:
```
Hive> CREATE TABLE jsonrow(str STRING);
Hive> LOAD  DATA LOCAL INPATH 'sample.json' INTO TABLE jsonrow;
Hive>CREATE TABLE target(a INT, b INT, c INT);
Hive> INSERT OVERWRITE INTO TABLE target
        SELECT get_json_object(str, $, 'a'), get_json_object(str, $, 'b'),  get_json_object(str, $, 'c')
        FROM jsonrow;
 Hive> SELECT * FROM target;
```
2nd model:
```
Hive> INSERT OVERWRITE TABLE target
        SELECT x.* FROM jsonrow
        lateral view json_tuple(str,a, b, c) x as a, b, c;
```
jsonfile.json

```
{"a":200, "b" : {"x":250, "y": 275}, "c" : 300}
{"a":210, "b" :{ "x":300, "y" : 350}, "c": 375}
Hive> CREATE TABLE jsontab;
Hive> LOAD DATA LOCAL INPATH 'jsonfile.json' OVERWRITE INTO TABLE jsontab;
Hive> INSERT OVERWRITE TABLE jsonsession
         SELECT x.* FROM jsonrow
         lateral view json-tuplestr, a,b, c)  x AS a, b, c;
Hive>CREATE TABLE jtarget(a int, bx int, by int, c int );
Hive> INSERT OVERWRITE TABLE jtarget
         SELECT a,x.*,c  FROM jsonsession;
         laterval view json_tuple(b, 'x', 'y') x AS bx, by;
```
Jsonfile.json:

```
{"cid": 250, "pr":[100, 200, 300]}
{"cid": 210, "pr":[100, 300]}
Hive> SELECT get_json_object(str, '$', "cid"), get_json_object(str, '$', "pr")[1]
        FROM jsonrow;
```
3rd real time model:
Download hive-json-serde.0.2.jar

```
hive> add jar  /home/training/desktop/hive-json-serde.o.2.jar
hive>CREATE TABLE target(a INT, b INT, c INT)
        ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde.jsonserde'
hive> LOAD DATA LOCAL INPATH 'jsonfile' INTO TABLE target;

{"a": 100, "b": 200, "c": 300}
{"a" : 50}
{"b":30}
{"c":40}
```

{“c” :45, “a” :55}
target :

A	B	C
100	200	300
50	Null	Null
Null	30	Null
Null	Null	40
55	Null	45

Hive> DESCRIBE target;
a INT with deserializer
b INT with deserializer
c INT with deserializer

Hive + HBase Integration:
Using this feature, we can process hbase table data using hql.
Hive> create table image(k string, a int, b int)
Stored by ‘org.apache.hadoop.hive.hbase.HBstoragehandler’
with SERDE PROPERTIES (‘hbase.columns.mapping’=
TBLPROPERTIES (“hbase.table.name’=’hbtab’);
Ok
$hbase shell
Hbase> list
parse-url-tuple: it is one of UDTS in hive which has to be applied with lateral view.
the basic parts of URL, -(parse-url-tuple) it returns host, path, query
syntax:
SELECT <alias> * FROM <table> lateral view parse-url-tuple
(<urlcolumn>,’<parameter>’, ‘<parameter2>’,-----) <alias> AS <columns to be guaranteed as lateral view>
example:
SELECT info.* FROM url lateral view parse-url-tuple()str, ‘HOST’ ‘PATH’, ‘QUERY’) INTO AS h,p,q;
str-to-map:
it converts a string into map collection.
str→’a=100*b=200’
str-to-map(str, ‘*’, ‘=’)
= → value delimiter
*→pair delimeter
[a:100, b:200]
m→[a:100, b:200]
to get the value of map collection
m[‘a’]→ returns 100
m[‘b’]→returns 200
→map_value(qmap)→array→[“101”,” 34“,”hyd”]
→map_keys(key)→array→[‘id’,’age’,’loc’]
hive>CREATE TABLE url(str STRING);
Hive>LOAD DATA LOCAL INPATH(sampleurl.txt’ INTO TABLE url;

```
Hive> CREATE TABLE urltable(h STRING, p STRING, q STRING);
Hive>INSERT OVERWRITE TABLE urltable
        SELECT info.* FROM url lateral view
        parse-url-view(str, ‘HOST’, ‘PATH’, ‘QUERY’) INTO AS h, p , q;
Hive> CREATE TABLE urlmap(h STRING, p  STRING), qmap MAP<STRING, STRING>);
Hive>INSERT OVERWRITE TABLE urlmap
         SELECT h, p, str-to-map(q, ‘&’, ‘=’) FROM urltab;
Hive> CREATE TABLE target (host STRING, path STRING, id STRING, age int, loc STRING);
Hive> >INSERT OVERWRITE TABLE target
         SELECT h, p, substr(p,2), qmap(‘id’), qmap(‘age’), qmap(‘loc’) FROM urlmap;
Hive>SELECT * FROM target;
```

size(qmap)→ it gives howmany number of map collections are there.

```
Hive> show function
Hive> DESCRIBE function name
Hive> DESCRIBE extended SUBSTR();

$ hive –f  filename.hql
```

How to do mass updates in hive:

Ac.no	Amount
------------ ------------	------------- -------------

```
Hive> insert overwrite table accounts
        Select Acno,  (Amount+Amount*0.2) FROM accounts
```