Hive

Contents

Hive

Hive is capable of only structured data.
Hive is a Hadoop-based data warehousing-like framework originally developed at Facebook.
It allows users to write queries in a SQL-like language called HiveQL(HQL), which are then converted to MapReduce jobs.
All Hadoop eco system components stores data on HDFS.
Hive is not case-sensitive
In Hive, everything will be in directories.
Default file format in Hive is text.
Default delimiter in Hive is ctrlA character. Its ASCII value is 001

Limitations of Hive

Hive doesn't provide record-level update, insert, nor delete.
Hadoop is batch-oriented system, hive queries have higher latency, due to start up overhead for MapReduce jobs.
Queries that would finish in seconds for a traditional database, takes longer for hive, even for relatively small datasets. Hive doesn't provide transactions.

Differences b/w RDBMS and Hive

We can perform update, insert on RDBMS
Hive stores the data on HDFS. So, we cannot change the data i.e., row level updates/deletes cannot be permitted.

OLTP – Online Transaction Processing Systems
Hive is not best suited for OLTP where continuous updates will be required.
Hive does not provide insert and update at row level. So, it is not suitable for OLTP system.

OLAP – Online Analytical Processing Systems
Hive is best suited for OLAP

RDBMS	HIVE
RDBMS is schema on write – which means while data is inserted it will perform lot of checks whether the values match with corresponding columns datatypes,	HIVE is schema on read – there will be no constraint checks

Hive doesn't support OLTP. If you need OLTP features, you should consider *NoSQL* databases like *HBase*, *Cassandra* and *Dynamo*DB, if you are using Amazon EMR / EC2

Hive can run on

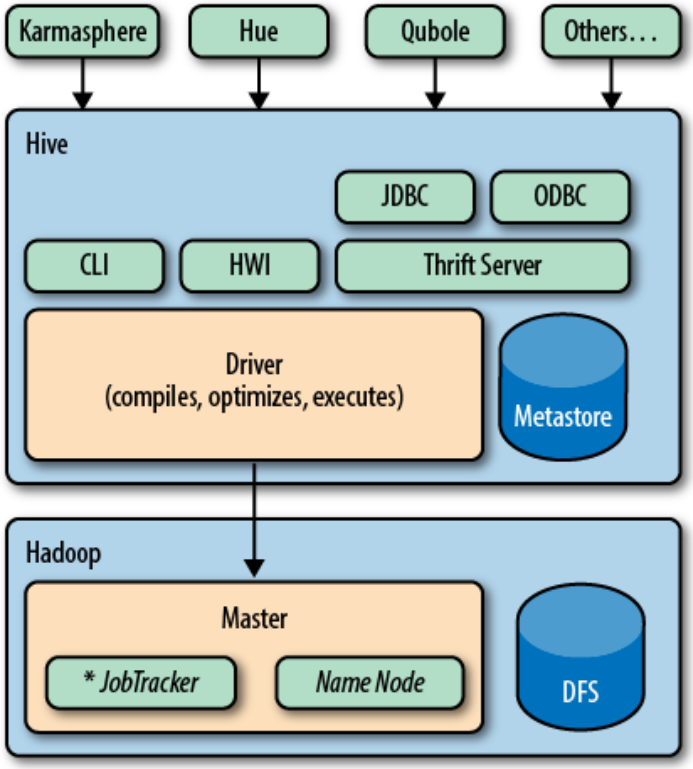
i) Local mode (Default)
ii) Distributed mode
iii)Pseudo-distributed mode

All Linux commands can be performed on hive using ! before the command.

```
hive> !pwd;
/home/training
hive> !whoami;
training
```

Hive

Hive Architecture



Here Karmasphere, Hue, Qubole are the various types to interact with hive.

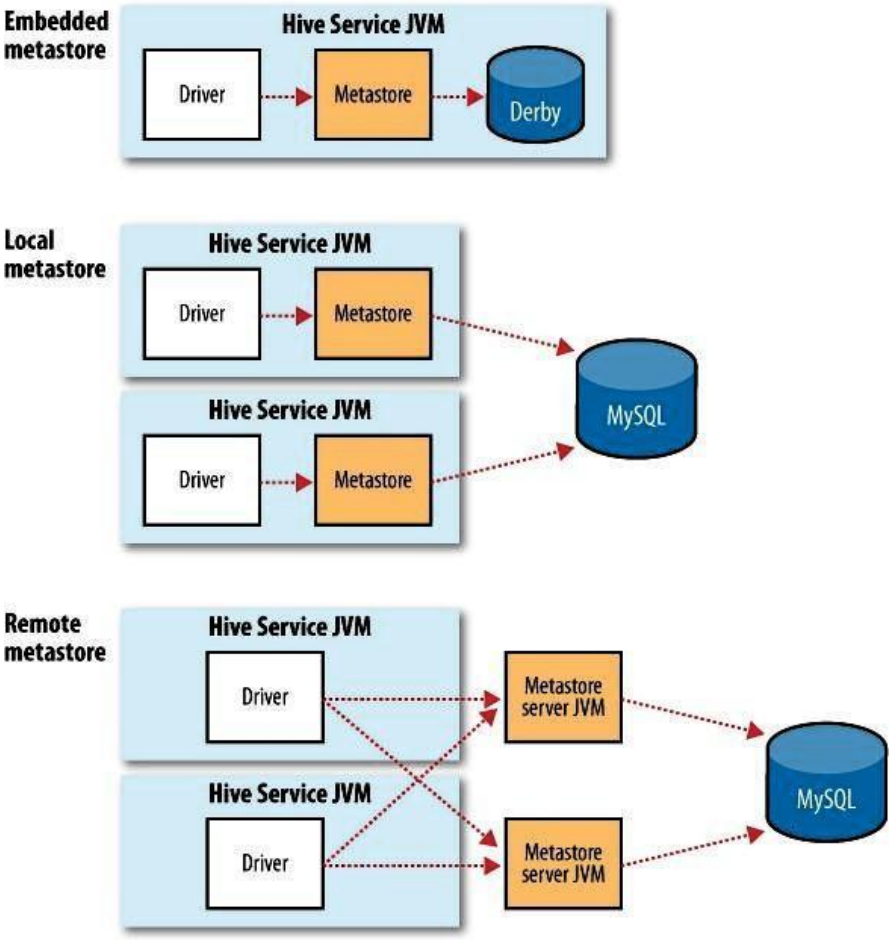
CLI – Command Line Interface (Terminal)
Web Interface – HUE editor on browser. We can connect hive, pig, sqoop, Oozie, flume
Thrift Server – used to validate the connections by make using of JDBC/ODBC

Compiler – compiles the java code into .class file
Optimizer –
Executor – Execute the java byte code

Metastore – It gives the entire information about the metadata. Metastore will be a RDBMS. It is externally maintained by default it is derby database. MySQL will be preferred in real time.

There are 3 types of Metastore.
Embedded metastore: Driver, Metastore and the database Derby will be running on single JVM
Local metastore: In this the database is decoupled and run separately.
Remote metastore: It is fully distributed each service is run on its own JVM

Hive



Hive communicates with the ***JobTracker*** to initiate the MapReduce jobs.

For Hive execution engine

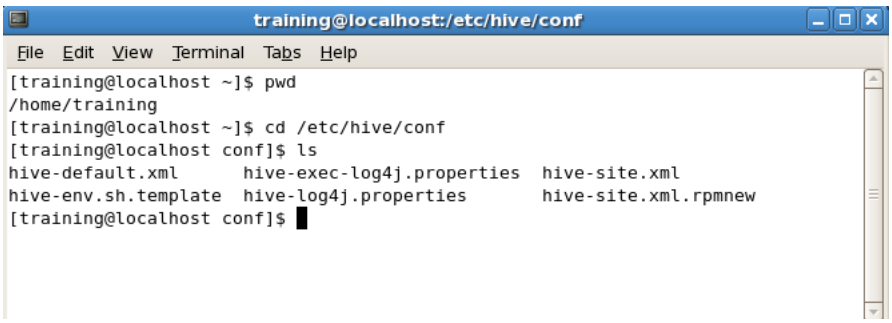
To change the execution engine to Tez use
SET hive.execution.engine=tez;
Tez is used in Hortonworks Sandbox

To change the execution engine to MapReduce use
SET hive.execution.engine=mr;

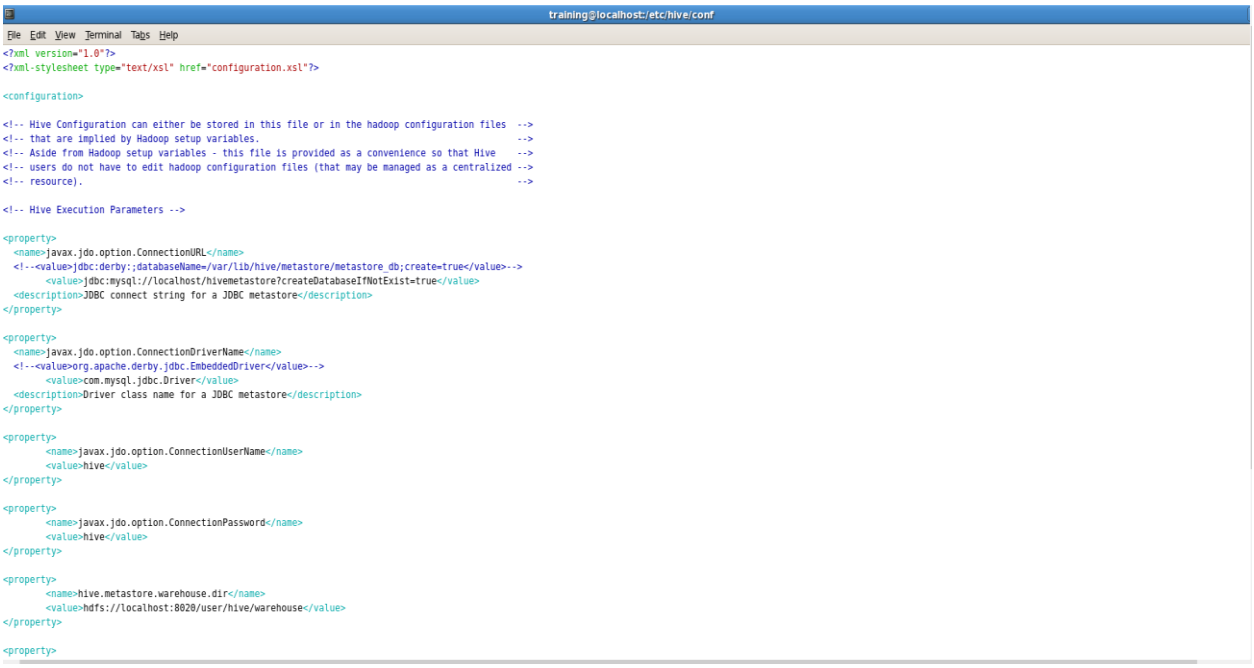
All configuration files related to hive will be present in

```
cd /etc /hive/conf
```

Hive

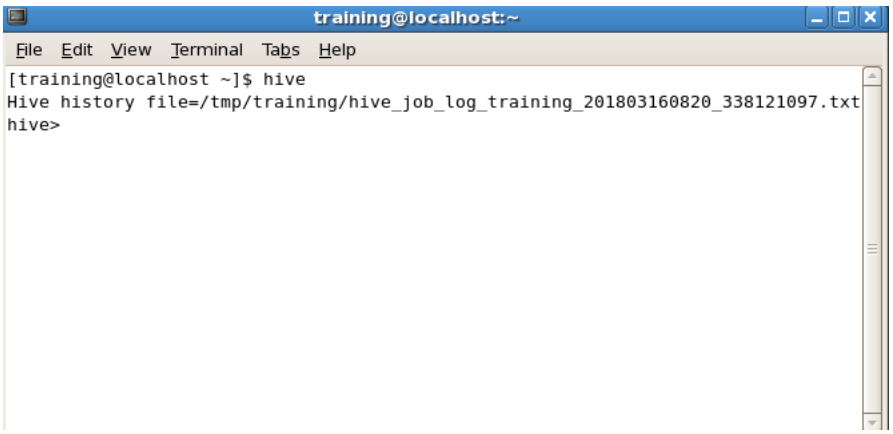


hive-site.xml



To start with hive, just type hive in the terminal

Hive



To list the databases, type

```
SHOW DATABASES;
```

To create the database, type

```
hive> CREATE DATABASE IF NOT EXISTS databasename
    > COMMENT 'database related to something';
```

The default database is default.

To enter the database, type

```
USE Databasename;
```

To lists the database starts with h, use

```
hive> SHOW DATABASES LIKE 'h.*';
```

A Hive extension is the ***RLIKE*** clause, which lets us use Java *regular expressions*, a more powerful minilanguage for specifying matches. The rich details of regular expression syntax and features are beyond the scope of this book. The entry for RLIKE in Table 6-6 provides links to resources with more details on regular expressions. Here, we demonstrate their use with an example, which finds all the employees whose street contains the word Chicago or Ontario:

```
hive> SELECT name, address.street
> FROM employees WHERE address.street RLIKE '.*(Chicago|Ontario).*';
```

Mary Smith 100 Ontario St.

Todd Jones 200 Chicago Ave.

The string after the RLIKE keyword has the following interpretation. A period (.) matches any character and a star (*) means repeat the "thing to the left" (period, in the two cases shown) zero to many times. The expression (x|y) means match *either* x *or* y.

To display the current database, type

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

To display the tables, type

```
SHOW TABLES;
```

Hive

```
hive -S  “SELECT * FROM tablename LIMIT 3”
```

Here ***-S Silent i.e., removes the OK and time taken***…. Lines as well as other inessential output

To view the commands history

```
$HOME/.hivehistory
```

If you are in hive terminal and wanted to execute the commands of LFS use **!**infornt of the command

```
hive> !pwd
```

Instead of ***hadoop fs -ls*** we can use ***dfs -ls***

To print the column headers, use

```
hive> SET hive.cli.print.header=true;
```

To create the table, type

```
CREATE TABLE tablename (attribute DATATYPE)
COMMENT ‘Description of the table’
TBLPROPERTIES (‘creator’=’nikhil’, ‘created_at’=’2018-23-03 10:00:00’, …);
```

TBLPROPERTIES can be used to express essential metadata about the database connection.

Copy the schema (but not the data) of an existing table, use

```
CREATE TABLE IF NOT EXISTS databasename.table2
LIKE databasename.table;
```

Ex:

```
CREATE TABLE IF NOT EXISTS mydb.employees2
LIKE mydb.employees;
```

If we aren’t in the same database, we can still list the tables in that database:

```
hive> USE default;
hive> SHOW TABLES IN mydb;
employees
```

Rename the table

```
ALTER TABLE oldtablename RENAME TO newtablename;
```

To load the data stored on HDFS, type

```
LOAD DATA INPATH ‘/hdfs path/file1.txt’ INTO TABLE tablename;
```

After the data is loaded, it will be disappeared from the original location

Hive

To load the data stored on LFS, type

```
LOAD DATA LOCAL INPATH '/LFS path/file1.txt' INTO TABLE tablename;
```

To display the table schema, type

```
DESCRIBE tablename;
```

To view the directory location of the database, use

```
DESCRIBE DATABASE databasename;
```

If you use local mode, the path will be *file:///user/hive/warehouse/database.db*.
Pseudo-distributed mode, the path will be *hdfs://master-slave/user/hive/warehouse/database.db*.
Ex hdfs://nn01.itversity.com:8020/apps/hive/warehouse/database.db
here nn01.itversity.com is the server name and 8020 is the port number
For a ***MapR*** installation, it would be ***maprfs***. For an
Amazon Elastic MapReduce (EMR) cluster, it would also be hdfs, but you could set
hive.metastore.warehouse.dir to use Amazon S3 explicitly (i.e., by specifying s3n://
bucketname/⋯ as the property value). You could use s3 as the scheme, but the newer
s3n is preferred

We can add properties to the database also, use

```
hive> CREATE DATABASE IF NOT EXISTS databasename
    > WITH DBPROPERTIES ('creator' = 'Mark Moneybags', 'date' = '2012-01-02');
```

To view the database properties, use

```
hive> DESCRIBE DATABASE EXTENDED databasename;
financials hdfs://master-server/user/hive/warehouse/ databasename.db
{date=2012-01-02, creator=Mark Moneybags);
```

By default, Hive won't permit you to drop a database if it contains tables. You can either
drop the tables first or append the **CASCADE** keyword to the command, which will cause
the Hive to drop the tables in the database first:

```
hive> DROP DATABASE IF EXISTS financials CASCADE;
```

Using the **RESTRICT** keyword instead of **CASCADE** is equivalent to the default behavior,
where existing tables must be dropped before dropping the database.
When a database is dropped, its directory is also deleted

To display the detailed table information, type

```
DESCRIBE EXTENDED tablename;
```

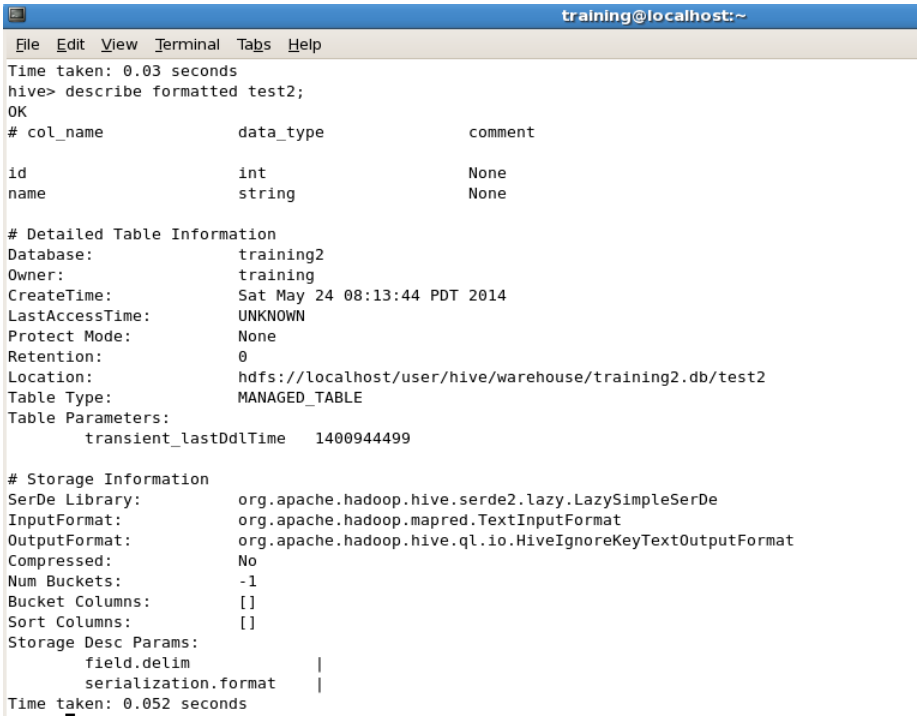
```
hive> describe extended test2;
OK
id      int
name    string

Detailed Table Information      Table(tableName:test2, dbName:training2, owner:training, createTime:1400944424, lastAccessTime:0, retentio
n:0, sd:StorageDescriptor(cols:[FieldSchema(name:id, type:int, comment:null), FieldSchema(name:name, type:string, comment:null)], location
:hdfs://localhost/user/hive/warehouse/training2.db/test2, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.ha
doop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache
.hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{serialization.format=|, field.delim=|}), bucketCols:[], sortCols:[], parameters:{}),
 partitionKeys:[], parameters:{transient_lastDdlTime=1400944499}, viewOriginalText:null, viewExpandedText:null, tableType:MANAGED_TABLE)
Time taken: 0.03 seconds
hive>
```

Hive

By default, input and output format will be text
to apply compression we need to change to true

To read it in human readable format

```
DESCRIBE FORMATTED tablename;
```



To display all the records of table,

```
SELECT * FROM tablename;
```

To display the limited number of records

```
SELECT * FROM tablename LIMIT 5;
```

Here 5 is the number of records to be displayed.

Real time

Once the table is created and data is loaded, the warehouse location is

```
/user/hive/warehouse/databasename.db/tablename/input.txt
```

Here databasename.db – is the database we have used
tablename – is the table we have created
input.txt – is the input file we have loaded into table

One way to edit the file on Hdfs, is to move it to LFS and edit the file
This is not recommended in real time, because it can contain GBs of data which is not possible to edit

Suppose we have the upstream, it daily produces some files. Then we use the above process
Then the daily produced file data will be appended to the existing data in the table.

To drop the table, use

Hive

```
DROP TABLE IF EXISTS tablename;
```

Datatypes in Hive

Primitive
Collection

Primitive data types

Type	*Size*	*Example*
TINYINT	1 byte signed integer	20
SMALLINT	2 byte signed integer	20
INT	4 byte signed integer	20
BIGINT	8 byte signed integer	20
BOOLEAN	Boolean true or false	TRUE
FLOAT	Single precision floating point	3.14159
DOUBLE	Double precision floating point	3.14159

Type	*Size*	*Example*
STRING	Sequence of characters. The character set can be specified. Single or double quotes can be used	'Now is the time', "for all good men"
TIMESTAMP (V0.8.0+)	Integer, float, or string	1327882394 (Unix epoch seconds), 1327882394.123456789 (Unix epoch seconds plus nanoseconds), and '2012-02-03 12:34:56.123456789' (JDBCcompliant java.sql.Timestamp format)
BINARY (V0.8.0+)	Array of bytes	

Values of the new TIMESTAMP type can be integers, which are interpreted as seconds since the Unix epoch time (Midnight, January 1, 1970), floats, which are interpreted as seconds since the epoch time with nanosecond resolution (up to 9 decimal places), and strings, which are interpreted according to the JDBC date string format convention, YYYY-MM-DD hh:mm:ss.fffffffff.

TIMESTAMPS are interpreted as UTC times. Built-in functions for conversion to and from timezones are provided by Hive, to_utc_timestamp and from_utc_timestamp, respectively

Type casting in Hive

where s is a string
column that holds a value representing an integer:

```
... cast(s AS INT) ...;
```

Collection data types

There are 3 collection data types in Hive.

- ARRAY
- MAP
- STRUCT

Hive

Most relational databases don't support such collection types, because using them tends to break *normal form*. For example, in traditional data models, structs might be captured in separate tables, with foreign key relations between the tables, as appropriate.

A practical problem with breaking normal form is the greater risk of data duplication, leading to unnecessary disk space consumption and potential data inconsistencies, as duplicate copies can grow out of sync as changes are made.

However, in *Big Data* systems, a benefit of sacrificing normal form is higher processing throughput. Scanning data off hard disks with minimal "head seeks" is essential when processing terabytes to petabytes of data. Embedding collections in records makes retrieval faster with minimal seeks. Navigating each foreign key relationship requires seeking across the disk, with significant performance overhead.

Text File Encoding of Data Values

Table 3-3. Hive's default record and field delimiters

Delimiter	Description
\n	For text files, each line is a record, so the line feed character separates records.
^A ("control" A)	Separates all fields (columns). Written using the octal code \001 when explicitly specified in CREATE TABLE statements.
^B	Separate the elements in an ARRAY or STRUCT, or the key-value pairs in a MAP. Written using the octal code \002 when explicitly specified in CREATE TABLE statements.
^C	Separate the key from the corresponding value in MAP key-value pairs. Written using the octal code \003 when explicitly specified in CREATE TABLE statements.

Schema on Read

When you write data to a traditional database, either through loading external data, writing the output of a query, doing UPDATE statements, etc., the database has total control over the storage. The database is the "gatekeeper." An important implication of this control is that the database can enforce the schema as data is *written*. This is called *schema on write*.

Hive has no such control over the underlying storage. There are many ways to create, modify, and even damage the data that Hive will query. Therefore, Hive can only enforce queries on *read*. This is called *schema on read*. So, what if the schema doesn't match the file contents? Hive does the best that it can to read the data. You will get lots of null values if there aren't enough fields in each record to match the schema. If some fields are numbers and Hive encounters non-numeric strings, it will return nulls for those fields. Above all else, Hive tries to recover from all errors as best it can.

Hive

HiveQL is the Hive Query Language.

Data Definition Language*(DDL)* parts of HiveQL, which are used for creating, altering, and dropping databases, tables, views, functions, and indexes.

Differences b/w EXTERNAL and MANAGED tables

Managed table	External table
Syntax CREATE TABLE tablename (attribute datatype) ROW FORMAT DELIMITED FIELDS TERMINATED BY 'specify delimiter' LINES TERMINATED BY '\n' STORED AS TEXTFILE; These are optional	CREATE EXTERNAL TABLE tablename (attribute datatype) ROW FORMAT DELIMITED FIELDS TERMINATED BY 'specify the delimiter' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
When to use If the table is limited to hive, use MANAGED table	If you want to create the table and access that table from Pig, Sqoop etc. It is better to use EXTERNAL TABLE
Functional difference When you drop a MANAGED table, all the underlying files residing on HDFS path will be deleted.	When you drop an EXTERNAL TABLE, all the underlying files residing on HDFS path will not be deleted.
When we drop a managed table, Hive deletes the data in the table.	Because it's external, Hive does not assume it *owns* the data. Therefore, dropping the table *does not* delete the data, although the *metadata* for the table will be deleted.

Default delimiter in Hive is ctrlA character. Its ASCII value is 001
HOW TO WRITE THE CTRLA CHARACTER
Press Ctrl+v a

To see the DDL statement of table, type

```
SHOW CREATE TABLE tablename;
```

PARTITIONING

Partitioning means dividing the table data into smaller parts based on the values of particular columns. Partitioning can be done on 1 or more columns. Partitioning are defined at the time of table creation using the **PARTITION BY** clause

Advantages

- 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.

Hive

For example,
we consider a table named Emp
it contains 15 million records
5m USA
5m EUR
5m CAN
select * from Emp where region='USA';

In RDBMS, if we run the query it must check entire 15m records. It takes much time and effects the efficiency.

To overcome this, Partitioning is introduced in Hive.

Disadvantages of Hive

Too many partitions in hive tables slow down the name node, storage space is wasted, Join queries become very slow.

WITHOUT PARTITIONING

hive> **CREATE TABLE nonpartitiontab (id INT, name STRING, region STRING)**
> **ROW FORMAT DELIMITED**
> **FIELDS TERMINATED BY ',';**

```
OK
Time taken: 0.051 seconds
```

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/input-allRegions.txt' INTO TABLE nonpartitiontab;**

```
Copying data from file:/home/training/Desktop/input-allRegions.txt
Copying file: file:/home/training/Desktop/input-allRegions.txt
Loading data to table hivedemo.nonpartitiontab
OK
Time taken: 0.154 seconds
```

hive> **SELECT * FROM nonpartitiontab;**

```
OK

123    anil    USA
111    anand   USA
222    dd      USA
333    ee      EUR
444    FF      EUR
555    GG      EUR
666    HHH     CAN
777    III     CAN
888    JJJ     CAN
Time taken: 0.11 seconds
```

hive> **SELECT * FROM nonpartitiontab WHERE region='USA';**

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803160749_0002, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201803160749_0002
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201803160749_0002
2018-03-16 12:11:00,712 Stage-1 map = 0%,  reduce = 0%
2018-03-16 12:11:01,724 Stage-1 map = 100%,  reduce = 0%
2018-03-16 12:11:03,737 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201803160749_0002
OK

123    anil    USA
111    anand   USA
222    dd      USA
Time taken: 5.965 seconds
```

Hive

WITH PARTITIONING

hive> **CREATE TABLE partitiontab(id INT, name STRING)**
> PARTITIONED BY (region STRING)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ',';

OK
Time taken: 0.058 seconds

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/input-usa.txt' INTO TABLE partitiontab PARTITION(region='USA');**

Copying data from file:/home/training/Desktop/input-usa.txt
Copying file: file:/home/training/Desktop/input-usa.txt
Loading data to table default.partitiontab partition (region=USA)
OK
Time taken: 0.297 seconds

hive> **SELECT * FROM partitiontab;**

OK
123 anil USA
111 anand USA
222 dd USA
NULL NULL USA

Time taken: 0.15 seconds

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/input-eur.txt' INTO TABLE partitiontab PARTITION(region='EUR');**

Copying data from file:/home/training/Desktop/input-eur.txt
Copying file: file:/home/training/Desktop/input-eur.txt
Loading data to table hivedemo.partitiontab partition (region=EUR)
OK
Time taken: 0.679 seconds

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/input-can.txt' INTO TABLE partitiontab PARTITION(region='CAN');**

Copying data from file:/home/training/Desktop/input-can.txt
Copying file: file:/home/training/Desktop/input-can.txt
Loading data to table hivedemo.partitiontab partition (region=CAN)
OK
Time taken: 0.182 seconds

hive> **SELECT * FROM partitiontab;**

OK

666 HHH CAN
777 III CAN
888 JJJ CAN
333 ee EUR
444 FF EUR
555 GG EUR
123 anil USA
111 anand USA
222 dd USA
NULL NULL USA
Time taken: 0.299 seconds

hive> **SELECT * FROM partitiontab where region='USA';**

OK

123 anil USA
111 anand USA
222 dd USA
NULL NULL USA
Time taken: 0.208 seconds

Without partitioning	With partitioning
hive> **SELECT * FROM nonpartitiontab WHERE region='USA';** Total MapReduce jobs = 1 Launching Job 1 out of 1 Number of reduce tasks is set to 0 since there's no reduce operator Starting Job = job_201803160749_0002, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201803160749_0002 Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill job_201803160749_0002 2018-03-16 12:11:00,712 Stage-1 map = 0%, reduce = 0%	hive> **SELECT * FROM partitiontab where region='USA';** OK 123 anil USA 111 anand USA 222 dd USA NULL NULL USA Time taken: 0.208 seconds

Hive

2018-03-16 12:11:01,724 Stage-1 map = 100%, reduce = 0%
2018-03-16 12:11:03,737 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201803160749_0002
OK
123 anil USA
111 anand USA
222 dd USA
Time taken: 5.965 seconds

It launches map reduce jobs because it has to search the entire data **/user/hive/warehouse/nonpartitiontab/input-allRegions.txt**

It doesn't launch map reduce jobs because it is partitioned by column named region.
Here column name is regions. So, data is separated by region. Only that particular data is present in that region.
/user/hive/warehouse/hivedemo.db/partitiontab/region=USA
/user/hive/warehouse/hivedemo.db/partitiontab/region=EUR
/user/hive/warehouse/hivedemo.db/partitiontab/region=CAN

hive> DESCRIBE FORMATTED nonpartitiontab;
OK
col_name data_type comment

id int None
name string None
region string None

Detailed Table Information
Database: default
Owner: training
CreateTime: Fri Mar 16 11:45:44 PDT 2018
LastAccessTime: UNKNOWN
Protect Mode: None
Retention: 0
Location: hdfs://localhost/user/hive/warehouse/nonpartitiontab
Table Type: MANAGED_TABLE
Table Parameters:
transient_lastDdlTime 1521226146

Storage Information
SerDe Library: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat: org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed: No
Num Buckets: -1
Bucket Columns: []
Sort Columns: []
Storage Desc Params:
field.delim ,
serialization.format ,
Time taken: 0.305 seconds

hive> DESCRIBE FORMATTED partitiontab;
OK
col_name data_type comment
id int None
name string None
Partition Information
col_name data_type comment
region string None

Detailed Table Information
Database: default
Owner: training
CreateTime: Fri Mar 16 11:56:24 PDT 2018
LastAccessTime: UNKNOWN
Protect Mode: None
Retention: 0
Location: hdfs://localhost/user/hive/warehouse/partitiontab
Table Type: MANAGED_TABLE
Table Parameters:
transient_lastDdlTime 1521226906

Storage Information
SerDe Library: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat: org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed: No
Num Buckets: -1
Bucket Columns: []
Sort Columns: []
Storage Desc Params:
field.delim ,
serialization.format ,
Time taken: 0.082 seconds

hadoop fs -ls hdfs://localhost/user/hive/warehouse/nonpartitiontab
Found 1 items
-rw-r--r-- 1 training supergroup 107 2018-03-16 11:49
/user/hive/warehouse/nonpartitiontab/input-allRegions.txt

hadoop fs -ls hdfs://localhost/user/hive/warehouse/hivedemo.db/partitiontab
Found 3 items
drwxr-xr-x - training supergroup 0 2018-03-16 12:18
/user/hive/warehouse/hivedemo.db/partitiontab/region=CAN
drwxr-xr-x - training supergroup 0 2018-03-16 12:18
/user/hive/warehouse/hivedemo.db/partitiontab/region=EUR
drwxr-xr-x - training supergroup 0 2018-03-16 12:12
/user/hive/warehouse/hivedemo.db/partitiontab/region=USA

hadoop fs -cat /user/hive/warehouse/nonpartitiontab/input-allRegions.txt
123,anil,USA
111,anand,USA
222,dd,USA
333,ee,EUR
444,FF,EUR
555,GG,EUR
666,HHH,CAN
777,III,CAN
888,JJJ,CAN

hadoop fs -ls /user/hive/warehouse/hivedemo.db/partitiontab/region=USA
Found 1 items
-rw-r--r-- 1 training supergroup 40 2018-03-16 12:12
/user/hive/warehouse/hivedemo.db/partitiontab/region=USA/input-usa.txt
hadoop fs -ls /user/hive/warehouse/hivedemo.db/partitiontab/region=EUR
Found 1 items
-rw-r--r-- 1 training supergroup 39 2018-03-16 12:18
/user/hive/warehouse/hivedemo.db/partitiontab/region=EUR/input-eur.txt
hadoop fs -ls /user/hive/warehouse/hivedemo.db/partitiontab/region=CAN
Found 1 items
-rw-r--r-- 1 training supergroup 35 2018-03-16 12:18
/user/hive/warehouse/hivedemo.db/partitiontab/region=CAN/input-can.txt

hadoop fs -cat /user/hive/warehouse/hivedemo.db/partitiontab/region=USA/input-usa.txt
123,anil
111,anand
222,dd
hadoop fs -cat /user/hive/warehouse/hivedemo.db/partitiontab/region=EUR/input-eur.txt
333,ee
444,FF

	555,GG **hadoop fs -cat /user/hive/warehouse/hivedemo.db/partitiontab/region=CAN/input-can.txt** 666,HHH 777,III 888,JJJ

Converting non-partitioning table -> partition table

Now you have a table named nonpartitiontab

hive> **SELECT * FROM nonpartitiontab;**
OK

123 anil USA
111 anand USA
222 dd USA
333 ee EUR
444 FF EUR
555 GG EUR
666 HHH CAN
777 III CAN
888 JJJ CAN
Time taken: 0.08 seconds

For example, in this we have only 3 regions. In real time, there could be any number of regions
i) We cannot directly convert non-partitioned table to partition table.
ii) We need to create new table, by taking the non-partitioned table as input

hive> **CREATE TABLE partitiontab2(id INT, name STRING)**
> **PARTITIONED BY (region STRING);**

OK
Time taken: 0.038 seconds
Here we don't need to explicitly mention ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' because we are trying to perform table to table

hive> **INSERT OVERWRITE TABLE partitiontab2 PARTITION(region='USA') SELECT t1.id,t1.name FROM nonpartitiontab t1 WHERE t1.region='USA';**

Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803160749_0003, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201803160749_0003
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill job_201803160749_0003
2018-03-16 13:21:38,617 Stage-1 map = 0%, reduce = 0%
2018-03-16 13:21:40,631 Stage-1 map = 100%, reduce = 0%

2018-03-16 13:21:41,641 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201803160749_0003
Ended Job = -2054466894, job is filtered out (removed at runtime).
Moving data to: hdfs://localhost/tmp/hive-training/hive_2018-03-16_13-21-33_782_7267372836280918174/-ext-10000
Loading data to table hivedemo.partitiontab2 partition (region=USA)
Partition hivedemo.partitiontab2{region=USA} stats: [num_files: 1, num_rows: 0, total_size: 26]
Table hivedemo.partitiontab2 stats: [num_partitions: 1, num_files: 1, num_rows: 0, total_size: 26]
3 Rows loaded to partitiontab2
OK
Time taken: 8.306 seconds

hive> **SELECT * FROM partitiontab2;**
OK

123 anil USA
111 anand USA
222 dd USA
Time taken: 0.19 seconds

Hive

hive> **INSERT OVERWRITE TABLE partitiontab2 PARTITION(region='EUR') SELECT t1.id,t1.name FROM nonpartitiontab t1 WHERE t1.region='EUR';**

```
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803160749_0004, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201803160749_0004
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201803160749_0004
2018-03-16 13:24:15,078 Stage-1 map = 0%,  reduce = 0%
2018-03-16 13:24:17,093 Stage-1 map = 100%,  reduce = 0%
2018-03-16 13:24:18,101 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201803160749_0004
Ended Job = -1343264562, job is filtered out (removed at runtime).
Moving data to: hdfs://localhost/tmp/hive-training/hive_2018-03-16_13-24-12_616_5286664750581306443/-ext-10000
Loading data to table hivedemo.partitiontab2 partition (region=USA)
Deleted hdfs://localhost/user/hive/warehouse/hivedemo.db/partitiontab2/region=USA
Partition hivedemo.partitiontab2{region=USA} stats: [num_files: 1, num_rows: 0, total_size: 21]
Table hivedemo.partitiontab2 stats: [num_partitions: 1, num_files: 1, num_rows: 0, total_size: 21]
3 Rows loaded to partitiontab2
OK
Time taken: 5.834 seconds
```

hive> **SELECT * FROM partitiontab2;**

```
OK

333    ee     EUR
444    FF     EUR
555    GG     EUR
123    anil   USA
111    anand  USA
222    dd     USA
```

hive> **INSERT OVERWRITE TABLE partitiontab2 PARTITION(region='CAN') SELECT t1.id,t1.name FROM nonpartitiontab t1 WHERE t1.region='CAN';**

```
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803160749_0005, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201803160749_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201803160749_0005
2018-03-16 13:25:42,914 Stage-1 map = 0%,  reduce = 0%
2018-03-16 13:25:44,929 Stage-1 map = 100%,  reduce = 0%
2018-03-16 13:25:45,938 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201803160749_0005
Ended Job = -342250526, job is filtered out (removed at runtime).
Moving data to: hdfs://localhost/tmp/hive-training/hive_2018-03-16_13-25-40_466_1690400796000185977/-ext-10000
Loading data to table hivedemo.partitiontab2 partition (region=USA)
Deleted hdfs://localhost/user/hive/warehouse/hivedemo.db/partitiontab2/region=USA
Partition hivedemo.partitiontab2{region=USA} stats: [num_files: 1, num_rows: 0, total_size: 24]
Table hivedemo.partitiontab2 stats: [num_partitions: 1, num_files: 1, num_rows: 0, total_size: 24]
3 Rows loaded to partitiontab2
OK
```

hive> **SELECT * FROM partitiontab2;**

```
OK

666    HHH    CAN
777    III    CAN
888    JJJ    CAN
333    ee     EUR
444    FF     EUR
555    GG     EUR
123    anil   USA
111    anand  USA
222    dd     USA
Time taken: 0.271 seconds
```

HOW WE DO IN REAL TIME

hive> **CREATE TABLE partitiontab3(id INT, name STRING)**
    > **PARTITIONED BY (region STRING);**

```
OK
Time taken: 0.024 seconds
```

Hive

//creation of tables will not be written in scripts

Now create a file named **vi hiveAutomation.hql**

use hivedemo;
INSERT OVERWRITE TABLE partitiontab3 PARTITION(region='USA') SELECT t1.id,t1.name FROM nonpartitiontab t1 WHERE t1.region='USA';
INSERT OVERWRITE TABLE partitiontab3 PARTITION(region='EUR') SELECT t1.id,t1.name FROM nonpartitiontab t1 WHERE t1.region='EUR';
INSERT OVERWRITE TABLE partitiontab3 PARTITION(region='CAN') SELECT t1.id,t1.name FROM nonpartitiontab t1 WHERE t1.region='CAN';

this is more of hard coding. We should not do this.

So, we should pass the region as the parameter. We have a parameter called **hive conf** which is a dynamic parameter

vi hiveAutomation.hql

use hivedemo;
INSERT OVERWRITE table partitiontab3 PARTITION (region='${hiveconf:region}') SELECT t1.id, t1.name FROM nonpartitiontab t1 WHERE t1.region='${hiveconf:region}';

Now run the script
hive -f hiveAutomation.hql -hiveconf region='USA'

Now open the hive terminal and run the select query
hive>SELECT * FROM partitiontab3;

123 anil USA
111 anand USA
222 dd USA

(Or)
You can pass the hiveconf parameter in a shell script

vi hiveAutomation.sh

echo "******started****"**
hive -f hiveAutomation.hql -hiveconf region='USA'
hive -f hiveAutomation.hql -hiveconf region='CAN'
hive -f hiveAutomation.hql -hiveconf region='EUR'
echo "****ended*****"**

sh hiveAutomation.sh **//**RUN THE SHELLSCRIPT

Now check the output
SELECT * FROM partitiontab3;
666 HHH CAN
777 III CAN
888 JJJ CAN
333 ee EUR
444 FF EUR
555 GG EUR
123 anil USA
111 anand USA
222 dd USA

What if there are 1000 regions we cannot invoke it by calling 1000 times. So, we use the concept of Dynamic Partitioning

MULTIPARTITIONINIG

In this, the partition column can have more than 1 column

hive> CREATE TABLE multipartition(id INT, name STRING)
> PARTITIONED BY (dt STRING, region STRING);

vi hiveMultipartition.hql

Hive

```
use hivedemo;
INSERT OVERWRITE table multipartition PARTITION (region='${hiveconf:region}', dt='${hiveconf:dat}' ) SELECT t1.id, t1.name FROM nonpartitiontab t1 WHERE
t1.region='${hiveconf:region}';
```

vi hiveMultipartition.sh

```
reg=$1       #1 is the argument
dateVar='date +%Y%m%d'
hive -f /home/training/oct2017/hiveMultipartitioning.hql -hiveconf region=$reg -hiveconf=$dateVar;
```

run the shellscript

```
sh -x hiveMultipartitioning.sh USA
```

check the metastore

```
hadoop fs -ls /user/hive/warehouse/oct2017.db/multipartition
/user/hive/warehouse/oct2017.db/multipartition/dt=20180316
/user/hive/warehouse/oct2017.db/multipartition/dt=20180316/region=USA
/user/hive/warehouse/oct2017.db/multipartition/dt=20180316/region=USA/000000_0
```

Dynamic partitioning

Instead of loading each partition with single SQL statement as shown above, which will result in writing lot of SQL statements for huge no of partitions, Hive supports dynamic partitioning with which we can add any number of partitions with single SQL execution.

Hive will automatically split our data into separate partition files based on the values of partition keys present in the input files.

if you have a lot of partitions to create, you must write a lot of SQL! Fortunately, hive also supports a *dynamic partition* feature, where it can infer the partitions to create based on query parameters
like
It gives the advantages of easy coding and no need of manual identification of partitions

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions.pernode=1000;
```

Static Partitioning	Dynamic Partitioning
• Insert input data files individually into a partition table is Static Partition.	• Single insert to partition table is known as a dynamic partition.
• Usually when loading files (big files) into **Hive tables** static partitions are preferred.	• Usually, dynamic partition loads the data from the non-partitioned table.
• Static Partition saves your time in loading data compared to dynamic partition.	• Dynamic Partition takes more time in loading data compared to static partition.
• You "statically" add a partition in the table and move the file into the partition of the table.	• When you have large data stored in a table then the Dynamic partition is suitable.

Hive

	• If you want to partition a number of columns but you don't know how many columns then also dynamic partition is suitable.
• We can alter the partition in the static partition.	• we can't perform alter on the Dynamic partition.
• You can get the partition column value from the filename, day of date etc without reading the whole big file.	

• If you want to use the Static partition in the hive you should set property **set hive.mapred.mode = strict** This property set by default in hive-site.xml	set hive.exec.dynamic.partition=true;
	set hive.exec.dynamic.partition.mode=nonstrict;
	set hive.exec.max.dynamic.partitions.pernode=1000;

• Static partition is in Strict Mode.	• Dynamic partition is in non-strict mode.
• You should use where clause to use limit in the static partition.	• Dynamic partition there is no required where clause to use limit.
• You can perform Static partition on Hive Manage table or external table.	• You can perform dynamic partition on hive external table and managed table.

BUCKETING(SUB PARTITIONING)

Hive partition divides table into number of partitions and these partitions can be further subdivided into more manageable parts known as **Buckets** or **Clusters**. The Bucketing concept is based on Hash function, which depends on the type of the bucketing column. Records which are bucketed by the same column will always be saved in the same bucket.

Here, ***CLUSTERED BY*** clause is used to divide the table into buckets.

In Hive Partition, each partition will be created as directory. But in Hive Buckets, each bucket will be created as file.

SET hive.enforce.bucketing=true;

Bucketing can also be done even without partitioning on Hive tables.

Hive

SORT MERGE BUCKET (SMB) JOIN

- In SMB join in Hive, each mapper reads a bucket from the first table and the corresponding bucket from the second table and then a merge sort join is performed.
- Sort Merge Bucket (SMB) join in hive is mainly used as there is no limit on file or partition or table join.
- SMB join can best be used when the tables are large.
- In SMB join the columns are bucketed and sorted using the join columns. All tables should have the same number of buckets in SMB join.

Complex Datatypes

array.txt
1,anil,60000,dccchc,pf#1500$vpf#2000,hyd$ap$500032
2,pavan,50000,dc$hc,pf#1300,bang$kar$600038

Here

dccchc are the types of cards. So, we use the concept called string. A String is a collection of items.
pf#1500$vpf#2000 are the pf and voluntary pf. They are in key-value pairs. So, we use map.

hive> **CREATE TABLE datatypestab(id INT, name STRING, salary BIGINT, cardtypes ARRAY<STRING>, pf MAP<STRING, INT>, address STRUCT<city:STRING, state:STRING, pincode:STRING>)**
> **ROW FORMAT DELIMITED**
> **FIELDS TERMINATED BY ','**
> **COLLECTION ITEMS TERMINATED BY '$'**
> **MAP KEYS TERMINATED BY '#';**

hive> LOAD DATA LOCAL INPATH '/home/training/Desktop/hiveinputs/array.txt' INTO TABLE datatypestab;

Copying data from file:/home/training/Desktop/hiveinputs/array.txt
Copying file: file:/home/training/Desktop/hiveinputs/array.txt
Loading data to table hivedemo.datatypestab
OK
Time taken: 0.306 seconds

hive> **SELECT * FROM datatypestab;**
OK

1 anil 60000 ["dc","cc","hc"] {"pf":1500,"vpf":2000} {"city":"hyd","state":"ap","pincode":"500032"}

2 pavan 50000 ["dc","hc"] {"pf":1300} {"city":"bang","state":"kar","pincode":"600038"}

Time taken: 0.122 seconds

hive> SELECT cardtypes[1] FROM datatypestab;

Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803190706_0001, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201803190706_0001
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill job_201803190706_0001|
2018-03-19 07:49:21,398 Stage-1 map = 0%, reduce = 0%
2018-03-19 07:49:23,420 Stage-1 map = 100%, reduce = 0%

Hive

2018-03-19 07:49:24,431 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201803190706_0001

cc
hc
Time taken: 7.708 seconds

hive> SELECT pf["vpf"] FROM datatypestab where id=1;

Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803190706_0002, Tracking URL =
http://localhost:50030/jobdetails.jsp?jobid=job_201803190706_0002
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill
job_201803190706_0002
2018-03-19 07:53:22,144 Stage-1 map = 0%, reduce = 0%
2018-03-19 07:53:24,161 Stage-1 map = 100%, reduce = 0%
2018-03-19 07:53:25,174 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201803190706_0002
OK
2000

In order to access the elements in struct, we use the period operator
hive> SELECT address.pincode FROM datatypestab WHERE id=2;

Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803190706_0005, Tracking URL =
http://localhost:50030/jobdetails.jsp?jobid=job_201803190706_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill
job_201803190706_0005
2018-03-19 07:57:22,463 Stage-1 map = 0%, reduce = 0%
2018-03-19 07:57:24,479 Stage-1 map = 100%, reduce = 0%
2018-03-19 07:57:25,487 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201803190706_0005
OK

600038
Time taken: 5.839 seconds

Table Syntax

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name    -- (Note:
TEMPORARY available in Hive 0.14.0 and later)
  [(col_name data_type [COMMENT col_comment], ... [constraint_specification])]
  [COMMENT table_comment]
```

Hive

```
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)] INTO
num_buckets BUCKETS]
  [SKEWED BY (col_name, col_name, ...)                  -- (Note: Available in
Hive 0.10.0 and later)]
     ON ((col_value, col_value, ...), (col_value, col_value, ...), ...)
     [STORED AS DIRECTORIES]
  [
   [ROW FORMAT row_format]
   [STORED AS file_format]
     | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]  -- (Note:
Available in Hive 0.6.0 and later)
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]   -- (Note: Available in
Hive 0.6.0 and later)
  [AS select_statement];   -- (Note: Available in Hive 0.5.0 and later; not
supported for external tables)

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  LIKE existing_table_or_view_name
  [LOCATION hdfs_path];

data_type
  : primitive_type
  | array_type
  | map_type
  | struct_type
  | union_type  -- (Note: Available in Hive 0.7.0 and later)

primitive_type
  : TINYINT
  | SMALLINT
  | INT
  | BIGINT
  | BOOLEAN
  | FLOAT
  | DOUBLE
  | DOUBLE PRECISION -- (Note: Available in Hive 2.2.0 and later)
  | STRING
  | BINARY      -- (Note: Available in Hive 0.8.0 and later)
  | TIMESTAMP   -- (Note: Available in Hive 0.8.0 and later)
  | DECIMAL     -- (Note: Available in Hive 0.11.0 and later)
  | DECIMAL(precision, scale)  -- (Note: Available in Hive 0.13.0 and later)
  | DATE        -- (Note: Available in Hive 0.12.0 and later)
  | VARCHAR     -- (Note: Available in Hive 0.12.0 and later)
  | CHAR        -- (Note: Available in Hive 0.13.0 and later)

array_type
  : ARRAY < data_type >

map_type
  : MAP < primitive_type, data_type >

struct_type
  : STRUCT < col_name : data_type [COMMENT col_comment], ...>

union_type
   : UNIONTYPE < data_type, data_type, ... >  -- (Note: Available in Hive 0.7.0 and
later)

row_format
```

Hive

```
  : DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS
TERMINATED BY char]
        [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
        [NULL DEFINED AS char]   -- (Note: Available in Hive 0.13 and later)
  | SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value,
property_name=property_value, ...)]

file_format:
  : SEQUENCEFILE
  | TEXTFILE    -- (Default, depending on hive.default.fileformat configuration)
  | RCFILE      -- (Note: Available in Hive 0.6.0 and later)
  | ORC         -- (Note: Available in Hive 0.11.0 and later)
  | PARQUET     -- (Note: Available in Hive 0.13.0 and later)
  | AVRO        -- (Note: Available in Hive 0.14.0 and later)
  | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

constraint_specification:
  : [, PRIMARY KEY (col_name, ...) DISABLE NOVALIDATE ]
    [, CONSTRAINT constraint_name FOREIGN KEY (col_name, ...) REFERENCES
table_name(col_name, ...) DISABLE NOVALIDATE
```

Joins : used to combine and retrieve the records from multiple tables

Inner Join

Left Join **:** returns all the values from left table + matched values from right table

Right Join : returns all the values from right table + matched values from left table

Left Outer Join **:** returns all the rows from left table, even if there are no matches in the right table

Right Outer Join: returns all the rows from right table, even if there are no matches in the left table

Full Outer Join **:** combines the records of both the left and the right outer tables that fulfil the JOIN condition

Map Side Join

Ex

Table 1	Table2	
100	2500000	records

50 ..Matching records

Now simply perform the join query on it

SELECT * FROM table1 t1 JOIN table2 t2 ON (t1.id=t2.id);

If we perform this query, it has to go through all the records

Here table1 is small and table2 is very big. Here comes the concept of **Map Side Join**

By default , if the size of the table is 25Mb we call it as smaller table.

Hive

property can be found in hive-site.xml
To enable this Map Side Join, we need to set the following property

```
SET hive.auto.convert.join=true;
```

In the Map Side Join, the smaller table is moved into cache called Distributed Cache.

Distributed Cache: is the cache directory which is same across all the slave nodes. It is an in-memory location.

How it works?
1. Scan the smallest table
2. Builds hash table on small table
(Hash table will be in key-value pair, here key will be the column selected(*) and value will be id in the above example)
3. Hash table generated will be uploaded to Distributed Cache

Example

smallTableInput.txt
123

bigTableInput.txt
123,anil
111,anand
222,DD
333,sharat
444,hari
555,abc
666,bbb
777,ccc
888,ddd
999,nnn
189,mnj
123,anil
111,anand
222,DD
333,sharat
444,hari
555,abc
666,bbb
777,ccc
888,ddd
999,nnn
189,mnj

hive> **CREATE TABLE smalltab(id INT);**
OK
Time taken: 0.084 seconds

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/hiveinputs/smallTableInput.txt' INTO TABLE smalltab;**

Copying data from file:/home/training/Desktop/hiveinputs/smallTableInput.txt
Copying file: file:/home/training/Desktop/hiveinputs/smallTableInput.txt

Hive

Loading data to table hivedemo.smalltab
OK
Time taken: 0.206 seconds

hive> **CREATE TABLE bigtab(id INT, name STRING);**
> **ROW FORMAT DELIMITED**
> FIELDS TERMINATED BY ',';
OK
Time taken: 0.04 seconds

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/hiveinputs/bigTableInput.txt' INTO TABLE bigtab;**

Copying data from file:/home/training/Desktop/hiveinputs/bigTableInput.txt
Copying file: file:/home/training/Desktop/hiveinputs/bigTableInput.txt
Loading data to table hivedemo.bigtab
OK
Time taken: 0.165 seconds

hive> **SET hive.auto.convert.join=true;**

hive> **SELECT * FROM smalltab t1 JOIN bigtab t2 ON (t1.id=t2.id);**

Total MapReduce jobs = 3
Ended Job = 2035335089, job is filtered out (removed at runtime).
Ended Job = -32064511, job is filtered out (removed at runtime).
Execution log at: /tmp/training/training_20180319083535_a7ea6df3-348f-4c69-86aa-24c6d5f8e73e.log
2018-03-19 08:35:13 Starting to launch local task to process map join; maximum memory = 202768384
2018-03-19 08:35:13 Processing rows: 1 Hashtable size: 1 Memory usage: 1674928 rate: 0.008
2018-03-19 08:35:13 Dump the hashtable into file: file:/tmp/training/hive_2018-03-19_08-35-11_016_2152015940337017459/-local-10004/HashTable-Stage-4/MapJoin-0--.hashtable
2018-03-19 08:35:13 Upload 1 File to: file:/tmp/training/hive_2018-03-19_08-35-11_016_2152015940337017459/-local-10004/HashTable-Stage-4/MapJoin-0--.hashtable File size: 342
2018-03-19 08:35:13 End of local task; Time Taken: 0.47 sec.
Mapred Local Task Succeeded . Convert the Join into MapJoin
Launching Job 2 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803190706_0006, Tracking URL =

http://localhost:50030/jobdetails.jsp?jobid=job_201803190706_0006|
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill job_201803190706_0006
2018-03-19 08:35:16,343 Stage-4 map = 0%, reduce = 0%
2018-03-19 08:35:18,357 Stage-4 map = 100%, reduce = 0%
2018-03-19 08:35:19,363 Stage-4 map = 100%, reduce = 100%
Ended Job = job_201803190706_0006
OK

123 123 anil
123 123 anil

Time taken: 8.444 seconds

If we perform normal inner join, the taken will be more than Map Side Join

==========

Hive

Till now, we have seen a text format file loading onto the hive table.
What if we want to load the JSON file, here comes the concept of SERDE
SERDE – Serializer and De-Serializer
- used to insert non-text format files for loading onto the hive table
Serializer – converting from **text format to JSON** format
De-Serializer – converting from **JSON to Text** format

Input file
simple.json
{"votes": {"abc":0, "def":0}, "type": "review","business id": "seggrdnnn","user_id": "eahrfsj","starts": "xbndxndm","text": "view"}

{"votes": {"abc":2, "def":6}, "type": "review","business id": "srdnnn","user_id": "ahrfsj","starts": "ndxndm","text": "view”}

Hive terminal

hive> **CREATE TABLE serdeJson(votes MAP<STRING, INT>,type STRING,business_id STRING, user_id STRING, starts STRING, text STRING)**
> **ROW FORMAT SERDE**
> 'com.cloudera.hive.serde.JSONSerDe'
> STORED AS TEXTFILE;

OK

hive> **add jar /usr/lib/hive/lib/hive-serdes-1.0-SNAPSHOT.jar;**

Added /usr/lib/hive/lib/hive-serdes-1.0-SNAPSHOT.jar to class path
Added resource: /usr/lib/hive/lib/hive-serdes-1.0-SNAPSHOT.jar

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/hiveinputs/sample.json' INTO TABLE serdeJson;**

Copying data from file:/home/training/Desktop/hiveinputs/sample.json
Copying file: file:/home/training/Desktop/hiveinputs/sample.json
Loading data to table hivedemo.serdejson
OK
Time taken: 0.16 seconds

hive> **SELECT * FROM serdeJson;**
OK

{"abc":0,"def":0} review NULL eahrfsj xbndxndm view
{"abc":2,"def":6} review NULL ahrfsj ndxndm view

Time taken: 0.043 seconds

Loading XML data into hive table using Serde

Hive

File Formats

TEXTFILE :

SEQUENCEFILE : if I have the plan to have the data in compressed manner, use sequence file.

ORC : It is the columnar format.

PARQUET : For impala, parquet is the default file format. The visual analytic people (tableau) access the hive tables using impala. The default file format in Impala is Parquet. It is the columnar formats.

AVRO :

When to go with ORC and Parquet?
If we have nested data, we go with Parquet

Views

A ***view*** allows a query to be saved and treated like a table. It is a logical construct, as it does not store data like a table. In other words, *materialized views* are not currently supported by Hive.

When a query references a view, the information in its definition is combined with the rest of the query by Hive's query planner. Logically, you can imagine that Hive executes the view and then uses the results in the rest of the query.

Views to Reduce Query Complexity

When a query becomes long or complicated, a view may be used to hide the complexity by dividing the query into smaller, more manageable pieces; similar to writing a function in a programming language or the concept of layered design in software. Encapsulating the complexity makes it easier for end users to construct complex queries from reusable parts.

HiveQL: Indexes

Hive has limited indexing capabilities. There are no keys in the usual relational database sense, but you can build an index on columns to speed some operations. The index data for a table is stored in another table.

Hive

Indexing is also a good alternative to partitioning when the logical partitions would actually, be too numerous and small to be useful. Indexing can aid in pruning some blocks from a table as input for a MapReduce job. Not all queries can benefit from an index.

RDBMS-

INDEX- table of content
- used for easy access of data

Types of INDEX in RDBMS
==========
BitMap : when data cardinality (uniqueness of data values in a table, Example gender will be male/female) is very low, we use BitMap
B-Tree (default) : when data cardinality is high (uniqueness of data values in a table example, id will have many)a, we use B-Tree
function based
reverse key based

Types of INDEX in HIVE
==========
BitMap
Compact: The **B-Tree in RDBMS** is called as **Compact in hive**

View – to represent an SQL query
- security feature

view is just storing the SQL statement
for example, we have a table with 3 columns id, name, region
client orders to restrict the name feature to the downstream
downstream wants to access the only the name id, region but not name

>CREATE VIEW secureview AS SELECT id, region FROM tablename WHERE id=555;
To create the view, type

```
CREATE VIEW viewname AS sql query;
```

Creating an index means creating a pointer on a particular column of a table

To create the index, type

```
CREATE INDEX indexname ON TABLE tablename (columnname) AS 'COMPACT' WITH DEFERRED REBUILD;
```

```
CREATE INDEX indexname ON TABLE tablename (columnname) AS 'BITMAP' WITH DEFERRED REBUILD;
```

If you specified WITH DEFERRED REBUILD, the new index starts empty. At any time, the index can be built the first time or rebuilt using the ALTER INDEX statement.

Showing an Index

```
SHOW FORMATTED INDEX ON tablename;
```

Dropping an Index

```
DROP INDEX IF EXISTS indexname ON TABLE tablename;
```

Tuning

The first step to learning how Hive works (after reading this book…) is to use the ***EXPLAIN*** feature to learn how Hive translates queries into MapReduce jobs.

EXPLAIN EXTENDED

```
hive> EXPLAIN select * from departments;
OK
STAGE DEPENDENCIES:
  Stage-0 is a root stage
STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        TableScan
          alias: departments
          Statistics: Num rows: 1 Data size: 90 Basic stats: COMPLETE Column stats: NONE
          Select Operator
            expressions: department_id (type: int), department_name (type: string)
            outputColumnNames: _col0, _col1
            Statistics: Num rows: 1 Data size: 90 Basic stats: COMPLETE Column stats: NONE
            ListSink
Time taken: 0.089 seconds, Fetched: 17 row(s)
```

```
hive> EXPLAIN EXTENDED select * from departments;
OK
ABSTRACT SYNTAX TREE:
TOK_QUERY
   TOK_FROM
      TOK_TABREF
         TOK_TABNAME
            departments
   TOK_INSERT
      TOK_DESTINATION
         TOK_DIR
            TOK_TMP_FILE
      TOK_SELECT
         TOK_SELEXPR
            TOK_ALLCOLREF
STAGE DEPENDENCIES:
  Stage-0 is a root stage
STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: -1
```

Hive

```
Processor Tree:
  TableScan
    alias: departments
    Statistics: Num rows: 1 Data size: 90 Basic stats: COMPLETE Column stats: NONE
    Select Operator
      expressions: department_id (type: int), department_name (type: string)
      outputColumnNames: _col0, _col1
      Statistics: Num rows: 1 Data size: 90 Basic stats: COMPLETE Column stats: NONE
      ListSink
      Time taken: 0.04 seconds, Fetched: 34 row(s)
```

Other File Formats and Compression

GZip Compression
BZip2 Compression
Snappy Compression
LZO Compression

BZip2 creates the smallest compressed output, but with the highest CPU overhead.
GZip is next in terms of compressed size versus speed. Hence, if disk space utilization and I/O overhead are concerns, both are attractive choices.
LZO and Snappy create larger files but are much faster, especially for decompression.
They are good choices if disk space and I/O overhead are less important than rapid decompression of frequently read data.

Functions

User-Defined Functions (UDFs) are a powerful feature that allow users to extend HiveQL.
HQL = HDFS + SQL (Queries data present on HDFS)

Functions

1. String manipulation functions
    - Case conversion: lower, upper, initcap (for camel case)
    - Concatenation: concat, ||
    - Extracting substring: substr, split(not available in oracle)
    - Indexing function: instr
    - Trimming functions: ltrim (left side/leading side), rtrim(right side/ trailing side, trim(on both sides)
    - Padding functions: lpad, rpad
    - Getting length: length
    - Replacing characters or substrings: regex_replace, translate
2. Type conversion
3. Date manipulations
    - Current date – current_date, current_timestamp, to_date
    - Arithmetic functions – date_add, date_sub, datediff, ,months_between
    - Extraction functions – year, month, date, next_day, trunc, last_day
    - Format functions – date_format
    - Unix timestamp to/from database timestamp conversion functions

to_unix_timestamp
unix_timestamp

4. Aggregate functions (Group by / over in select)

To lists the functions currently loaded in the Hive session, use

SHOW FUNCTIONS;
DESCRIBE FUNCTION function name;
DESCRIBE FUNCTION EXTENDED function name;

Ex: **describe function substr;**

```
hive (nikhil_poc)> select substr("Hello World", 1, 5);
OK
Hello
Time taken: 1.171 seconds, Fetched: 1 row(s)
```

```
hive (nikhil_poc)> select count(1) from orders01;
OK
68883
Time taken: 0.836 seconds, Fetched: 1 row(s)
```

String manipulations

```
hive (nikhil_poc)> select lower(order_status) from orders01 limit 5;
OK
closed
pending_payment
complete
closed
complete
Time taken: 0.439 seconds, Fetched: 5 row(s)

hive (nikhil_poc)> select upper(order_status) from orders01 limit 5;
OK
CLOSED
PENDING_PAYMENT
COMPLETE
CLOSED
COMPLETE
Time taken: 0.226 seconds, Fetched: 5 row(s)
```

substr()

```
hive (nikhil_poc)> select substr("Hello World", 3, 3);
OK
llo
Time taken: 0.619 seconds, Fetched: 1 row(s)
```

Here 3,3
The first 3 is the position from where you want to extract
The second 3 is the number of characters you want to extract

Typically, in programming language, we do it in the following way

"Hello World".split(2,5)
Here 2 is the index we are starting

And 5 is the length of the string upto what we want

hive (nikhil_poc)> **select substr(order_date, 1, 10) from orders01 limit 2;**
OK
2013-07-25
2013-07-25
Time taken: 1.05 seconds, Fetched: 2 row(s)

hive (nikhil_poc)> **select split(order_date, ',') from orders01 limit 2;**
OK
["2013-07-25 00:00:00"]
["2013-07-25 00:00:00"]
Time taken: 0.221 seconds, Fetched: 2 row(s)

hive (nikhil_poc)> **select split(order_date, ' ') from orders01 limit 2;**
OK
["2013-07-25","00:00:00"]
["2013-07-25","00:00:00"]
Time taken: 0.212 seconds, Fetched: 2 row(s)

hive (nikhil_poc)> **select split(order_date, ' ')[0] from orders01 limit 2;**
OK
2013-07-25
2013-07-25
Time taken: 0.271 seconds, Fetched: 2 row(s)

instr(str, substr) - Returns the index of the first occurance of substr in str

Displays the index of 00

hive (nikhil_poc)> **select instr(order_date, '00') from orders01 limit 2;**
OK
12
12
Time taken: 0.215 seconds, Fetched: 2 row(s)

ltrim(str) - Removes the leading space characters from str
rtrim(str) - Removes the trailing space characters from str

Ex:
“ Hello World0000”
We want to trim from beginning as well as ending

hive (nikhil_poc)> **select ltrim(" Hello World0000");**
OK
Hello World0000
Time taken: 1.28 seconds, Fetched: 1 row(s)

To check whether it is trimmed or not

select trim(order_date) from orders01 where length(trim(order_date)) != length(order_date);

Padding and concat

Ex
Year month date
2017 2 3

2017 10 1

hive (nikhil_poc)> select 2017, 2, 3;
OK
2017 2 3
Time taken: 0.417 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select concat(2017, '-', '2', '-', '3');**
OK
2017-2-3
Time taken: 0.273 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select concat(2017, '-', '02', '-', '03');**
OK
2017-02-03
Time taken: 0.312 seconds, Fetched: 1 row(s)

Realtime

If your date is represented as integers in the database, and want to display appropriately in the front end we use **lpad**

hive (nikhil_poc)> **select concat(2017, '-', lpad(2,2,0), '-', lpad(3,2,0));**
OK
2017-02-03
Time taken: 0.351 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select * from orders LIMIT 5;**
OK
1 2013-07-25 00:00:00 11599 CLOSED
2 2013-07-25 00:00:00 256 PENDING_PAYMENT
3 2013-07-25 00:00:00 12111 COMPLETE
4 2013-07-25 00:00:00 8827 CLOSED
5 2013-07-25 00:00:00 11318 COMPLETE
Time taken: 0.307 seconds, Fetched: 5 row(s)

Now we want to replace – with /

First, we must split the column on which we are performing
Then use regex_replace

hive (nikhil_poc)> **select split(order_date, " ")[0] from orders limit 3;**
OK
2013-07-25
2013-07-25
2013-07-25
Time taken: 0.241 seconds, Fetched: 3 row(s)

hive (nikhil_poc)> **select regexp_replace(split(order_date, " ")[0], '-', '/') from orders limit 3;**
OK
2013/07/25
2013/07/25
2013/07/25
Time taken: 0.275 seconds, Fetched: 3 row(s)

If we want to replace multiple characters,
Typically, we use regexp_replace many times, but to be more easier way, we use translate

Hive

```
hive (nikhil_poc)> select translate("Hello World", "lo", "12");
OK
He112 W2r1d
```

Type conversion

```
hive (nikhil_poc)> select cast("2013" as int);
OK
2013
Time taken: 0.672 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> select cast(substr(order_date, 1, 4) as int) from orders limit 1;
OK
2013

1,4 means character starting from 1 and ending upto 4 character

hive (nikhil_poc)> select * from orders where cast(substr(order_date, 1, 4) as int) = 2013 limit 1;
OK
1       2013-07-25 00:00:00     11599   CLOSED
Time taken: 0.438 seconds, Fetched: 1 row(s)
```

Date manipulations

```
select current_date;
select current_timestamp;

to extract the date column (we can use split or to_date functions)
select to_date(order_date) from orders limit 5;

we want to get the data of last 4 days

hive (nikhil_poc)> select * from orders where to_date(order_date) between date_add(current_date, -4) and
date_add(current_date, -1);
OK
Time taken: 0.341 seconds

hive (nikhil_poc)> select date_add(current_date, -4), date_add(current_date, -1);
OK
2018-06-22      2018-06-25
Time taken: 0.254 seconds, Fetched: 1 row(s)

We can write the above query as the following way also
hive (nikhil_poc)> select datediff("2018-06-26", "2018-06-22");
OK
4
Time taken: 0.617 seconds, Fetched: 1 row(s)
```

Hive

To see the difference between months, we use the following

hive (nikhil_poc)> **select months_between("2018-06-26", "2018-06-22");**
OK
0.12903226
Time taken: 0.329 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select year(current_date);**
OK
2018
Time taken: 0.283 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select month(current_date);**
OK
6
Time taken: 0.266 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select day(current_date);**
OK
26
Time taken: 0.261 seconds, Fetched: 1 row(s)

trunc is primarily used to get year-to-date and month-to-date reports as part of where clause

hive (nikhil_poc)> **select trunc(current_date, 'MM'); //gives the first date of month**
OK
2018-06-01
Time taken: 0.293 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select trunc(current_date, 'YY'); //gives the first date of the year**
OK
2018-01-01
Time taken: 0.312 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> **select last_day(current_date); // gives the ;ast date of the year**
OK
2018-06-30
Time taken: 0.279 seconds, Fetched: 1 row(s)

select date_format(order_date, 'YY') from orders limit 10;
select date_format(order_date, 'YYYY') from orders limit 10;

hive (nikhil_poc)> select unix_timestamp(current_date);
OK
1529985600
Time taken: 0.341 seconds, Fetched: 1 row(s)

hive (nikhil_poc)> select unix_timestamp(current_timestamp);

Hive

Convert unix timestamp to actual date

select to_unix_timestamp(order_date, 'YYYY-MM') from orders limit 10;

Standard Functions

Mathematical functions
round()
floor()
abs()

String functions

ucase()
reverse()
concat()

Aggregate Functions

sum()
min()
max()
avg()
count()

User Defined Functions (UDFs) in Hive

In the above table data,
hive> **SELECT * FROM bigtab;**
OK

111,anand
222,DD
333,sharat
444,hari
555,abc
666,bbb
777,ccc
888,ddd
999,nnn
189,mnj
123,anil
111,anand
222,DD
333,sharat
444,hari
555,abc

Hive

666,bbb
777,ccc
888,ddd
999,nnn
189,mnj

Time taken: 0.134 seconds

WHAT IF WE WANTED TO CONVERT INTO UPPERCASE, WE HAVE A FUNCTION AVAILABLE IN HIVE CALLED UPPER. IF WE WANT TO USE THE CUSTOMIZED FUNCTION, WE WRITE THE UDFs.

hive> **SELECT UPPER(name) FROM bigtab;**

Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803190706_0007, Tracking URL =
http://localhost:50030/jobdetails.jsp?jobid=job_201803190706_0007
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill
job_201803190706_0007
2018-03-19 08:46:37,439 Stage-1 map = 0%, reduce = 0%
2018-03-19 08:46:39,449 Stage-1 map = 100%, reduce = 0%

2018-03-19 08:46:40,457 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201803190706_0007
OK

ANIL
ANAND
DD
SHARAT
HARI
ABC
BBB
CCC
DDD
NNN
MNJ
ANIL
ANAND
DD
SHARAT
HARI
ABC
BBB
CCC
DDD
NNN
MNJ
Time taken: 5.506 seconds

Hive

Java UDF

Input file

HiveJUDF.txt
abc|cde
nikhil|Vemula
hadoop|analytics

Here we are writing a udf to replace the string

UdfHIVETest.java

```
package org.abc.cde;
import org.apache.hadoop.hive.ql.exec.UDF;
public class UdfHIVETest extends UDF {
        private Text result = new Text();

        public Text evaluate (String str, String matchPattern, String replacePattern) {

                String rep = str.replace(matchPattern, replacePattern);
                result.set(rep);
                return result; //return new Text(rep);
        }

}
```

```
hive> CREATE TABLE udftest(fname STRING, lname STRING)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '|';
OK
Time taken: 0.043 seconds

hive> LOAD DATA LOCAL INPATH '/home/training/Desktop/hiveinputs/HiveJUDF.txt' INTO TABLE udftest;

Copying data from file:/home/training/Desktop/hiveinputs/HiveJUDF.txt
Copying file: file:/home/training/Desktop/hiveinputs/HiveJUDF.txt
Loading data to table hivedemo.udftest
OK
Time taken: 0.175 seconds

hive> SELECT * FROM udftest;
OK

abc     cde
nikhil  Vemula
hadoop  analytics
```

Hive

Time taken: 0.109 seconds

Now replace hadoop with bigdata

Now export the UdfHIVETest.java as jar file
Export->JAR file->/home/training/Desktop/testHiveUDF.jar

The jar is in LFS
Now you want to utilize the jar in hive
extend the scope of this jar to all the datanodes (this is where the distributed cache comes to picture)
Add the jar to the distributed cache

hive>**add jar /home/training/Desktop/testHiveUDF.jar;**
Added /home/training/Desktop/testHiveUDF.jar to the class path
Added resource: /home/training/Desktop/testHiveUDF.jar

hive> **CREATE TEMPORARY FUNCTION replaceword AS "org.abc.cde.UdfHIVETEST";**
hive> SELECT repalceword(fname,'hadoop','bigdata'), lname FROM udftest;
output
abc cde
nikhil Vemula
bigdata analytics

==========

Python UDF

vi MaxTransaction.py

```
#!/usr/bin/python

import sys #sending input from keyboard

def findMostTransactionType(healthcard, travelcard, debitcard, creditcard, petrolcard):

  cardTypes = {0 : 'healthcard', 1 : 'travelcard', 2 : 'debitcard', 3 : 'creditcard', 4 : 'petrolcard'}

  cards = [healthcard, travelcard, debitcard, creditcard, petrolcard]

  maxValue = max(cards)

  return cardTypes[cards.index(maxValue)]


for line in sys.stdin:

  line = line.strip()

  a, b, c, d, e = line.split('\t')

  print findMostTransactionType(a, b, c, d, e, acctNo)
```

Hive

Hive terminal

hive> **CREATE TABLE maxtranc(id INT, hc INT, tc INT, dc INT, CC int, PC int)**
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ',';

OK
Time taken: 0.031 seconds

hive> **LOAD DATA LOCAL INPATH '/home/training/Desktop/hiveinputs/HivePyUDF.txt' INTO TABLE maxtranc;**

Copying data from file:/home/training/Desktop/hiveinputs/HivePyUDF.txt
Copying file: file:/home/training/Desktop/hiveinputs/HivePyUDF.txt
Loading data to table hivedemo.maxtranc
OK
Time taken: 0.47 seconds

hive> **SELECT * FROM maxtranc;**
OK

11111111 5 2 4 6 7

22222222 6 7 8 1 2

33333333 2 5 1 9 3

Time taken: 0.039 seconds

Bring the scope of the python code

Now distribute the python code across the cluster (slave nodes) i.e., bring it to the Distributed Cache

hive> **add file /home/training/workingdir/python/MaxTransaction.py;**
Added resource: /home/training/workingdir/python/MaxTransaction.py

hive> **SELECT TRANSFORM(hc,tc,dc,cc,id) USING 'python MaxTransaction.py' as Maxtransc FROM maxtranc;**
OK
('111', 'petrolcard')
('222', 'debitcard')
('333', 'creditcard')
Time taken: 7.44 seconds

hive> SELECT * FROM nonpartitiontab;
OK

123 anil USA
111 anand USA
222 dd USA
333 ee EUR
444 FF EUR
555 GG EUR

Hive

666 HHH CAN
777 III CAN
888 JJJ CAN
Time taken: 0.192 seconds

hive> CREATE INDEX testindex ON TABLE nonpartitiontab (id) AS 'COMPACT' WITH DEFERRED REBUILD;

OK
Time taken: 0.149 seconds

hive> **SELECT * FROM nonpartitiontab where id=555;**

Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201803190706_0008, Tracking URL =
http://localhost:50030/jobdetails.jsp?jobid=job_201803190706_0008
Kill Command = /usr/lib/hadoop/bin/hadoop job -Dmapred.job.tracker=localhost:8021 -kill job_201803190706_0008
2018-03-19 09:54:10,450 Stage-1 map = 0%, reduce = 0%
2018-03-19 09:54:12,471 Stage-1 map = 100%, reduce = 0%
2018-03-19 09:54:13,480 Stage-1 map = 100%, reduce = 100%

Ended Job = job_201803190706_0008
OK

555 GG EUR

Time taken: 5.532 seconds

Before creating the view, when you perform the same the select query it takes much time when compared the query after creating the view.

Streaming

Streaming offers an alternative way to transform data. During a streaming job, the Hadoop Streaming API opens an I/O pipe to an external process. Data is then passed to the process, which operates on the data it reads from the *standard input* and writes the results out through the *standard output*, and back to the Streaming API job. While Hive does not leverage the Hadoop streaming API directly, it works in a very similar way.

Hive provides several clauses to use streaming:
MAP()
REDUCE()
TRANSFORM()

Hive

Customizing Hive File and Record Formats

```
file_format:
  : SEQUENCEFILE
  | TEXTFILE    -- (Default, depending on hive.default.fileformat configuration)
  | RCFILE      -- (Note: Available in Hive 0.6.0 and later)
  | ORC         -- (Note: Available in Hive 0.11.0 and later)
  | PARQUET     -- (Note: Available in Hive 0.13.0 and later)
  | AVRO        -- (Note: Available in Hive 0.14.0 and later)
  | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname
```

Sequence files are flat files consisting of binary key-value pairs. When Hive converts queries to MapReduce jobs, it decides on the appropriate key-value pairs to use for a given record.

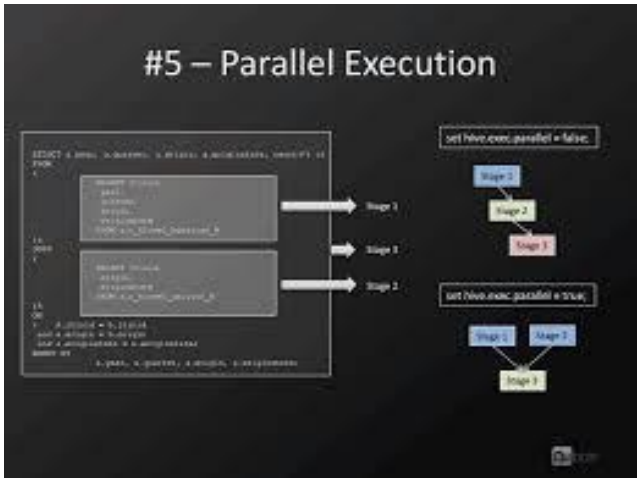
Another efficient binary format that is supported natively by Hive is RCFile.

* RCFile stores columns of a table in a record columnar way. It first partitions rows horizontally into row splits. and then it vertically partitions each row split in a columnar way.

RCFile first stores the meta data of a row split, as the key part of a record, and all the data of a row split as the value part.

Optimization techniques

1. Partitioning
   - Static Partitioning
   - Dynamic Partitioning
2. Bucketing
3. Parallel execution
   - SET hive.exec.parallel=true



4. Optimize Joins
5. File Formats: ORC, PARQUET, AVRO
6. Hive Indexing
7. Use Vectorization

Hive

Vectorized query execution improves the performance of operations like scans, aggregations, filters and joins, by performing them in batches of 1024 rows at once instead of single row each time.

Introduced in Hive 0.13, this feature significantly improves query execution time, and is easily enabled with 2 parameter settings.

SET hive.vectorized.execution.enabled=true;
SET hive.vectorized.execution.reduce.enabled=true;

8. Cost Based Query Optimization

SET hive.cbo.enable=true;
SET hive.compute.query.using.stats=true;
SET hive.stats.fetch.column.stats=true;
SET hive.stats.fetch.partition.stats=true;
And then within Hive, enter the command
ANALYZE TABLE orc_table COMPUTE STATISTICS;