Hive

There are two ways to connect to hive

CLI

We have two CLIs
---
1) Hive

```
[rishabhtiwari2048gmail@ip-10-0-41-79 ~]$ hive
WARNING: Use "yarn jar" to launch YARN applications.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/jars/log4j-slf4j-impl-2.8.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/jars/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Logging initialized using configuration in jar:file:/opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/jars/hive-common-2.1.1-cdh6.3.2.jar!/hive-log4j2.properties Async: false
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive>
```

2) Beeline - Since hive is deprecated, we must use beeline to connect to hive from CLI

```
[rishabhtiwari2048gmail@ip-10-0-41-79 ~]$ beeline
WARNING: Use "yarn jar" to launch YARN applications.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/jars/log4j-slf4j-impl-2.8.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/jars/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Beeline version 2.1.1-cdh6.3.2 by Apache Hive
beeline> !connect jdbc:hive2://10.0.21.22:10000
Connecting to jdbc:hive2://10.0.21.22:10000
Enter username for jdbc:hive2://10.0.21.22:10000: rishabhtiwari2048gmail
Enter password for jdbc:hive2://10.0.21.22:10000: **************************
Connected to: Apache Hive (version 2.1.1-cdh6.3.2)
Driver: Hive JDBC (version 2.1.1-cdh6.3.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://10.0.21.22:10000>
```

Editor

We can also use hive editor provided in Hue by Cloudera

Note: like SQL hive is case insensitive language

Create database in hive

0: jdbc:hive2://10.0.21.22:10000> create database rishabh2;

Drop database in hive

Note: You have to put cascade in order to drop a database if database is not empty (i.e. database contains tables) oth erwise cascade is optional

0: jdbc:hive2://10.0.21.22:10000> drop rishabhdb cascade;

See the list of databases in hive

0: jdbc:hive2://10.0.21.22:10000> show databases;

Connect to a specific database in hive

0: jdbc:hive2://10.0.21.22:10000> use rishabhdb;

Note: In hive CLI you can use following property in order to display the name of the database to which you are curr ently connected

hive>set hive.cli.print.current.db = true;
hive (rishabhdb)>

See the list of tables in specific database

0: jdbc:hive2://10.0.21.22:10000> show tables;

Create managed table

Managed tables are Hive owned tables where the entire lifecycle of the tables' data are managed and controlled by H ive. This means that data associated with the table is directly controlled by hive and if table is dropped the data assoc iated with that table will also be deleted from HDFS.

0: jdbc:hive2://10.0.21.22:10000> create table flights (year INT, month INT, airline STRING, distance INT, source STRING, destination STRING, departure INT, arrival INT, delay INT, diverted INT, canceled INT) row format deli mited fields terminated by ',' lines terminated by '\n';

Load the data into table from HDFS location.

0: jdbc:hive2://10.0.21.22:10000> LOAD DATA INPATH '/user/rishabhtiwari2048gmail/hive/flights.csv' INTO TA BLE flights;

Note:- Now if you go and see the location in HDFS where this file was present, you won't find it there because hive

have moved this file into /user/hive/warehouse/rishabhdb.db/flights/flights.csv. Now if you drop this table the flights .csv present in this location will also be dropped.

Load the data into table from local file system

Note: You can also use file present on your local file system to populate data into hive tables.

0: jdbc:hive2://10.0.21.22:10000> create table airlines (iata_code STRING, name STRING) row format delimited fi elds terminated by ',' lines terminated by ',';

hive (rishabhdb)> load data local inpath '/mnt/home/rishabhtiwari2048gmail/airlines.csv' into table airlines;

Query the managed hive table to see its content

0: jdbc:hive2://10.0.21.22:10000> select * from airlines limit 5;

See the table structure of a hive table

0: jdbc:hive2://10.0.21.22:10000> describe flights;

```
+--------------+------------+----------+
|   col_name   | data_type  | comment  |
+--------------+------------+----------+
| year         | int        |          |
| month        | int        |          |
| airline      | string     |          |
| distance     | int        |          |
| source       | string     |          |
| destination  | string     |          |
| departure    | int        |          |
| arrival      | int        |          |
| delay        | int        |          |
| diverted     | int        |          |
| canceled     | int        |          |
+--------------+------------+----------+
```

See more details of a table

0: jdbc:hive2://10.0.21.22:10000> describe formatted flights;

```
+-------------------------------+----------------------------------------------------+-----------------------+
|           col_name            |                     data_type                      |        comment        |
+-------------------------------+----------------------------------------------------+-----------------------+
| # col_name                    | data_type                                          | comment               |
|                               | NULL                                               | NULL                  |
| year                          | int                                                |                       |
| month                         | int                                                |                       |
| airline                       | string                                             |                       |
| distance                      | int                                                |                       |
| source                        | string                                             |                       |
```

```
| destination                 | string                                             |                       |
| departure                   | int                                                |                       |
| arrival                     | int                                                |                       |
| delay                       | int                                                |                       |
| diverted                    | int                                                |                       |
| canceled                    | int                                                |                       |
|                             | NULL                                               | NULL                  |
| # Detailed Table Information  | NULL                                             | NULL                  |
| Database:                   | rishabhdb                                          | NULL                  |
| OwnerType:                  | USER                                               | NULL                  |
| Owner:                      | rishabhtiwari2048gmail                             | NULL                  |
| CreateTime:                 | Mon Apr 05 03:03:03 UTC 2021                       | NULL                  |
| LastAccessTime:             | UNKNOWN                                            | NULL                  |
| Retention:                  | 0                                                  | NULL                  |
| Location:                   | hdfs://nameservice1/user/hive/warehouse/rishabhdb.db/flights | NULL        |
| Table Type:                 | MANAGED_TABLE                                      | NULL                  |
| Table Parameters:           | NULL                                               | NULL                  |
|                             | numFiles                                           | 1                     |
|                             | numRows                                            | 0                     |
|                             | rawDataSize                                        | 0                     |
|                             | totalSize                                          | 41608947              |
|                             | transient_lastDdlTime                              | 1617591862            |
|                             | NULL                                               | NULL                  |
| # Storage Information       | NULL                                               | NULL                  |
| SerDe Library:              | org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe | NULL                  |
| InputFormat:                | org.apache.hadoop.mapred.TextInputFormat           | NULL                  |
| OutputFormat:               | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat | NULL          |
| Compressed:                 | No                                                 | NULL                  |
| Num Buckets:                | -1                                                 | NULL                  |
| Bucket Columns:             | []                                                 | NULL                  |
| Sort Columns:               | []                                                 | NULL                  |
| Storage Desc Params:        | NULL                                               | NULL                  |
|                             | field.delim                                        | ,                     |
|                             | line.delim                                         | \n                    |
|                             | serialization.format                               | ,                     |
+-------------------------------+----------------------------------------------------+-----------------------+
```

Creating external tables in hive

An external table is a table for which Hive does not manage storage. If you delete an external table, only the definiti on in Hive is deleted. The data remains.

There are three ways to create external tables in hive
----

1) Create external table and insert data directly

0: jdbc:hive2://10.0.21.22:10000> CREATE EXTERNAL TABLE employee_data (eid INT, ename STRING, salary INT, department STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATE D BY '\n' LOCATION '/user/rishabhtiwari2048gmail/hive/employee';

0: jdbc:hive2://10.0.21.22:10000> INSERT INTO employee_data VALUES (100,'Rishabh',150000,'Analytics');

2) Create external table and load data from local file system
---

CREATE EXTERNAL TABLE awards (name STRING, event STRING, year INT, category STRING, outcome ST
RING, language STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED
BY '\n' LOCATION '/user/rishabhtiwari2048gmail/hive/awards';

LOAD DATA LOCAL INPATH '/mnt/home/rishabhtiwari2048gmail/Hive/data2/awards_data.csv' INTO TABLE a
wards;

LOAD DATA LOCAL INPATH '/mnt/home/rishabhtiwari2048gmail/Hive/data2/awards_data.csv' OVERWRITE I
NTO TABLE awards;

3) Create external table and load data from HDFS

CREATE EXTERNAL TABLE awards (name STRING, event STRING, year INT, category STRING, outcome ST
RING, language STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED
BY '\n' LOCATION '/user/rishabhtiwari2048gmail/hive/awards';

LOAD DATA INPATH '/mnt/home/rishabhtiwari2048gmail/Hive/data2/awards_data.csv' INTO TABLE awards;

Create an external table in hive

0: jdbc:hive2://10.0.21.22:10000> create external table timesheet (driverid INT, week INT, hours_logged INT, miles
_logged INT) row format delimited fields terminated by ',' lines terminated by '\n' tblproperties("skip.header.line.cou
nt"="1");

Load data in hive table

0: jdbc:hive2://10.0.21.22:10000> load data inpath '/user/rishabhtiwari2048gmail/hive/hive-project-2/timesheet.csv'
into table timesheet;

Select data from hive table

0: jdbc:hive2://10.0.21.22:10000> select * from timesheet limit 5;

```
+--------------------+-----------------+------------------------+------------------------+
| timesheet.driverid  | timesheet.week  | timesheet.hours_logged  | timesheet.miles_logged  |
+--------------------+-----------------+------------------------+------------------------+
| 10                 | 1               | 70                     | 3300                   |
| 10                 | 2               | 70                     | 3300                   |
```

```
| 10                | 3              | 60                    | 2800                  |
| 10                | 4              | 70                    | 3100                  |
| 10                | 5              | 70                    | 3200                  |
+-------------------+----------------+-----------------------+-----------------------+
```

Dropping a table in hive

```
0: jdbc:hive2://10.0.21.22:10000> drop table timesheet;
```

Using serde classes to parse schema in different file formats
---

A SerDe is a short name for a Serializer Deserializer. Hive uses SerDe to read and write data from tables.

Create a table using Regex serde

```
0: jdbc:hive2://10.0.21.22:10000> create external table student_data (student_name STRING, student_id INT, stude
nt_fees INT) row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties ("input.regex" = "([A-Za-z]+) ([0-9]{3}) ([0-9]{4})") stored as textfile tblproperties("skip.head
er.line.count"="1");
```

Load data in hive table

```
0: jdbc:hive2://10.0.21.22:10000> load data inpath '/user/rishabhtiwari2048gmail/hive/log/sample.txt' into table stud
ent_data;
```

Hit hive table to see the loaded data

```
0: jdbc:hive2://10.0.21.22:10000> select * from student_data;
```

HCatalog

HCatalog is a hive sub-project that provides other tools of bigdata ecosystem to access Hive metastore

```
[rishabhtiwari2048gmail@ip-10-0-41-79 ~]$ pig -useHCatalog

grunt> mydata = LOAD 'rishabhdb.student_data' USING org.apache.hive.hcatalog.pig.HCatLoader();

grunt> dump mydata;
```

Creating Hive tables using Sqoop

[rishabhtiwari2048gmail@ip-10-0-41-79 ~]$ sqoop import --connect jdbc:mysql://sqoopdb.slbdh.cloudlabs.com/rishabhtiwari2048gmail --driver com.mysql.jdbc.Driver --username rishabhtiwari2048gmail -P --table rishabhdb.airline_safety -m 1 --hive-import

Creating hive table using Avro data format

We can use avro-tools to find schema in the avro documents

avro-tools getschema FlumeData.1617680158913 > schema.avsc

To know more about avro data structures there is a beautiful article on https://docs.oracle.com/database/nosql-12.1.3.0/GettingStartedGuide/avroschemas.html#avro-complexdatatypes

CREATE TABLE twitter_data ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED as INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat' OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat' TBLPROPERTIES ('avro.schema.url'='/user/rishabhtiwari2048gmail/hive_avro/schema.avsc');

0: jdbc:hive2://10.0.21.22:10000> load data inpath '/user/rishabhtiwari2048gmail/hive_avro/Data' into table rishabhdb.twitter_data;

SELECT * FROM twitter_data LIMIT 10;

Creating hive table using Parquet data format

You can convert your csv file using python's pandas and pyarrow library into a parquet file.

flights.csv size = 40 MB
flights.parquet size = 8 MB 80% Compression achieved

get the schema of flights.parquet using following command

```
[rishabhtiwari2048gmail@ip-10-0-41-79 p_data]$ parquet-tools schema flights.parquet
message schema {
  optional int64 year;
  optional int64 month;
  optional binary airline (STRING);
  optional int64 distance;
  optional binary source (STRING);
  optional binary destinaton (STRING);
  optional int64 arrival;
  optional double departure;
  optional double delay;
  optional int64 diverted;
  optional int64 canceled;
}
```

Note: Note the datatypes of columns in the parquet schema. Use datatypes while creating hive tables accordingly

hive (rishabhdb)> create table flights_2 (year BIGINT, month BIGINT, airline STRING, distance BIGINT, source STRING, destination STRING, arrival BIGINT, departure DOUBLE, delay DOUBLE,diverted BIGINT, canceled BI

GINT) row format delimited stored as parquet location '/user/rishabhtiwari2048gmail/hive';

0: jdbc:hive2://10.0.21.22:10000> select * from flights_2 limit 5;
INFO : Compiling command(queryId=hive_20210409161941_07f0861e-bc01-4ca2-99e1-2f75b85097ae): select * f
rom flights_2 limit 5
INFO : Semantic Analysis Completed
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:flights_2.year, type:bigint, comment:nu
ll), FieldSchema(name:flights_2.month, type:bigint, comment:null), FieldSche
ma(name:flights_2.airline, type:string, comment:null), FieldSchema(name:flights_2.distance, type:bigint, comment:
null), FieldSchema(name:flights_2.source, type:string, comment:null), F
ieldSchema(name:flights_2.destination, type:string, comment:null), FieldSchema(name:flights_2.arrival, type:bigint,
 comment:null), FieldSchema(name:flights_2.departure, type:double, co
mment:null), FieldSchema(name:flights_2.delay, type:double, comment:null), FieldSchema(name:flights_2.diverted
, type:bigint, comment:null), FieldSchema(name:flights_2.canceled, type:bi
gint, comment:null)], properties:null)
INFO : Completed compiling command(queryId=hive_20210409161941_07f0861e-bc01-4ca2-99e1-2f75b85097ae
); Time taken: 0.034 seconds
INFO : Executing command(queryId=hive_20210409161941_07f0861e-bc01-4ca2-99e1-2f75b85097ae): select * fr
om flights_2 limit 5
INFO : Completed executing command(queryId=hive_20210409161941_07f0861e-bc01-4ca2-99e1-2f75b85097ae)
; Time taken: 0.0 seconds
INFO : OK
+----------------+-----------------+-------------------+--------------------+------------------+-----------------------+---------
----------+---------------------+-------------
----+--------------------+--------------------+
| flights_2.year | flights_2.month | flights_2.airline | flights_2.distance | flights_2.source | flights_2.destination | fl
ights_2.arrival | flights_2.departure | flights_2.del
ay | flights_2.diverted | flights_2.canceled |
+----------------+-----------------+-------------------+--------------------+------------------+-----------------------+---------
----------+---------------------+-------------
----+--------------------+--------------------+
| 2015 | 1 | AS | 98 | ANC | NULL | 1448 | 408.0
         | -22.0
   | 0 | 0 |
| 2015 | 1 | AA | 2336 | LAX | NULL | 2330 | 741.0
           | -9.0
   | 0 | 0 |
| 2015 | 1 | US | 840 | SFO | NULL | 2296 | 811.0
         | 5.0
   | 0 | 0 |
| 2015 | 1 | AA | 258 | LAX | NULL | 2342 | 756.0
          | -9.0
   | 0 | 0 |
| 2015 | 1 | AS | 135 | SEA | NULL | 1448 | 259.0
         | -21.0
   | 0 | 0 |
+----------------+-----------------+-------------------+--------------------+------------------+-----------------------+---------
----------+---------------------+-------------
----+--------------------+--------------------+

Paritioning in hive

Steps:

1) Create hive table

```
hive (rishabhdb)> create external table flights (year int, month int, airline string, distance int, source string, destinati
on string, departure int, arrival int, delay int, diverted int, canceled int) row format delimited fields terminated by ',' l
ines terminated by '\n' location '/user/rishabhtiwari2048gmail/Apr13/hive1';
```

2) Load data in hive table

```
hive (rishabhdb)> load data local inpath '/mnt/home/rishabhtiwari2048gmail/Hive/flights.csv' into table flights;
```

3) Create partition table

```
hive (rishabhdb)> create table flights_part (month int, airline string, distance int, source string, destination string, de
parture int, arrival int, delay int, diverted int, canceled int) partitioned by (year string) row format delimited fields ter
minated by ',' lines terminated by '\n' stored as textfile location '/user/rishabhtiwari2048gmail/Apr13/partition/hive1';
```

4) Set hive.exec.dynamic.partition.mode=nonstrict

```
hive (rishabhdb)> set hive.exec.dynamic.partition.mode=nonstrict;
```

5) Load data from hive table into partition table

```
hive (rishabhdb)> insert overwrite table flights_part partition(year) select * from flights;
```

IMPORTANT: Partitioning is always done based on columns while Bucketting is done based on rows
---

```
0: jdbc:hive2://10.0.21.22:10000> create table postoffice (name string, type string, pincode bigint, contact string, tal
uka string, district string, divison string, division_headoffice string, headoffice string, state string) row format delimi
ted stored as parquet location '/user/rishabhtiwari2048gmail/hive';

0: jdbc:hive2://10.0.21.22:10000> SELECT * FROM postoffice WHERE state = 'MAHARASHTRA';

Time taken 7,004 rows selected (19.798 seconds)

0: jdbc:hive2://10.0.21.22:10000> select count(*) from postoffice where state = 'MAHARASHTRA';

Time taken 1 row selected (19.995 seconds)

+-------+
|  _c0  |
+-------+
| 7004  |
+-------+
```

Dynamic Partitioning

with partition

0: jdbc:hive2://10.0.21.22:10000> create external table postoffice2 (name string, type string, pincode bigint, contact string, taluka string, district string, divison string, division_headoffice string, headoffice string) partitioned by (state string) row format delimited fields terminated by ',' lines terminated by '\n' stored as parquet;

0: jdbc:hive2://10.0.21.22:10000> insert overwrite table postoffice2 partition (state) select * from postoffice;

0: jdbc:hive2://10.0.21.22:10000> select * from postoffice2 where state = 'MAHARASHTRA';

7,004 rows selected (4.482 seconds)

0: jdbc:hive2://10.0.21.22:10000> SELECT count(*) FROM postoffice2 WHERE state = 'MAHARASHTRA';

```
+-------+
|  _c0  |
+-------+
| 7004  |
+-------+
```
1 row selected (20.059 seconds)

Static Partitioning

table will be created in same way but while putting data in table, you will explicitly define table partition

0: jdbc:hive2://10.0.21.22:10000> create external table postoffice3 (name string, type string, pincode bigint, contact string, taluka string, district string, divison string, division_headoffice string, headoffice string) partitioned by (state string) row format delimited fields terminated by ',' lines terminated by '\n' stored as parquet;

0: jdbc:hive2://10.0.21.22:10000> insert into postoffice3 partition(state='PUNJAB') SELECT name, type, pincode, t aluka, district, division, division_headoffice, headoffice FROM postoffice WHERE state='PUNJAB';

0: jdbc:hive2://10.0.21.22:10000> select * from postoffice3 limit 5;

0: jdbc:hive2://10.0.21.22:10000> alter table postoffice3 drop partition (state='PUNJAB');

Bucketing

```
create external table postoffice4 (
name string,
type string,
pincode bigint,
contact string,
taluka string,
district string,
divison string,
division_headoffice string,
headoffice string,
state string)
clustered by (pincode) into 10 buckets
row format delimited
fields terminated by ','
lines terminated by '\n'
stored as parquet;
```

insert overwrite table postoffice4 select * from postoffice;

select * from postoffice4 limit 20;

Bucketting in Hive
---
1) Create hive table
---
hive (rishabhdb)> create external table flights (year int, month int, airline string, distance int, source string, destinati
on string, departure int, arrival int, delay int, diverted int, canceled int) row format delimited fields terminated by ',' l
ines terminated by '\n' location '/user/rishabhtiwari2048gmail/Apr13/hive1';

2) Load data in hive table
---
hive (rishabhdb)> load data local inpath '/mnt/home/rishabhtiwari2048gmail/Hive/flights.csv' into table flights;

3)Create bucket table
---
hive (rishabhdb)> create table flights_buck (year int, month int, airline string, distance int, source string, destination
string, departure int, arrival int, delay int, diverted int, canceled int) clustered by (airline) sorted by (year, month) int
o 5 buckets row format delimited fields terminated by ',' lines terminated by '\n' stored as parquet location '/user/rish
abhtiwari2048gmail/Apr13/partition/hive_bucket';

4)Set hive.enforce.bucketing=true
---
hive (rishabhdb)> set hive.enforce.bucketing=true;

5)Insert data in bucket table
---
hive (rishabhdb)> insert overwrite table flights_buck select * from flights;

6)We can also pull data only of a specific bucket
---
hive (rishabhdb)> select * from flights_buck tablesample(bucket 2 out of 5 on month);

Hive built in functions

CONCAT: to concat to strings

0: jdbc:hive2://10.0.21.22:10000> select concat('Rishabh','Tiwari') from dual;

Note: here dual is a dummy table that I created to work with built in functions. its mandatory to use a table while cal
ling builtin functions in select statement in Hive.

SUBSTR: to get a substring out of parent string

0: jdbc:hive2://10.0.21.22:10000> select substr('Rishabh',2,3) from dual;

+------+
| _c0 |

```
+------+
| ish  |
+------+
1 row selected (0.078 seconds)
```

UPPER/UCASE: Returns uppercase version of a string

```
0: jdbc:hive2://10.0.21.22:10000> select name,upper(name) from directors_data limit 3;
0: jdbc:hive2://10.0.21.22:10000> select name,ucase(name) from directors_data limit 3;
```

LOWER/LCASE: Returns lowercase version of a string

```
0: jdbc:hive2://10.0.21.22:10000> SELECT name, lower(name) FROM directors_data LIMIT 2;
0: jdbc:hive2://10.0.21.22:10000> SELECT name, lcase(name) FROM directors_data LIMIT 2;
```

TRIM/LTRIM/RTRIM: Returns string after removing whitespaces from corresponding side of a string

---

```
0: jdbc:hive2://10.0.21.22:10000> select trim('    Welcome    ') from dual;
0: jdbc:hive2://10.0.21.22:10000> select ltrim('    Welcome    ') from dual;
0: jdbc:hive2://10.0.21.22:10000> select rtrim('    Welcome    ') from dual;
```

CURRENT_DATE : Gives current system date.

```
0: jdbc:hive2://10.0.21.22:10000> SELECT CURRENT_DATE from dual;
```

YEAR(),MONTH(),DAY(): Gives corresponding components of a date

---

```
0: jdbc:hive2://10.0.21.22:10000> select year(CURRENT_DATE) FROM dual;
0: jdbc:hive2://10.0.21.22:10000> select month(CURRENT_DATE) FROM dual;
0: jdbc:hive2://10.0.21.22:10000> select day(CURRENT_DATE) FROM dual;
```

Aggregration Functions

Aggregation functions available for group of columns.

count(*) -- Count number of records present in the table

```
0: jdbc:hive2://10.0.21.22:10000> select count(*) from flights;
```

count(exp) -- Count number of entries in a particular column

```
0: jdbc:hive2://10.0.21.22:10000> select count(year) from flights;
```

sum(exp) -- calculate sum of exp provided

```
0: jdbc:hive2://10.0.21.22:10000> select airline, sum(diverted) from flights group by airline;
```

sum(distinct exp) -- eliminates duplicate values while calculating sum

avg(exp) - calculates mean
min(exp) - minimum of a group
max(exp) - maximum of a group

Joins in Hive

HiveQL Select Joins

Inner Join

0: jdbc:hive2://10.0.21.22:10000> SELECT a.year, a.month, b.name, a.source, a.destination FROM flights a INNER JOIN airlines b ON (a.airline = b.IATA_code) LIMIT 5;

Right Outer Join

0: jdbc:hive2://10.0.21.22:10000> SELECT a.year, a.month, b.name, a.source, a.destination FROM flights a RIGHT OUTER JOIN airlines b ON (a.airline = b.IATA_code) LIMIT 5;

Left Outer Join

0: jdbc:hive2://10.0.21.22:10000> SELECT a.year, a.month, b.name, a.source, a.destination FROM flights a RIGHT OUTER JOIN airlines b ON (a.airline = b.IATA_code) LIMIT 5;

Full Outer Join

0: jdbc:hive2://10.0.21.22:10000> SELECT a.year, a.month, b.name, a.source, a.destination FROM flights a FULL OUTER JOIN airlines b ON (a.airline = b.IATA_code) LIMIT 5;

Map Join in Hive

Normal HiveQL joins trigger Map and Reduce operations in the background whenever a query with Joins in it is fire d. However, in scenarios where one table is so small that its entire dataset
can be stored in memory of the data nodes while MapReduce operation, map join can be used to speed up the execut ion. In Map Joins, there are no reducers. All the join operations are done by
mapper only.
We can trigger Map Join in hive by two ways

1) Setting following properties before firing queries in hive
---
set hive.auto.convert.join = true
set hive.mapjoin.smalltable.filesize (default value of this property is 25 MB)

2) Using /*+MAPJOIN(<table name>)*/ hint
---
hive> SELECT /*+MAPJOIN(b)*/a.year, a.month, b.name, a.source, a.destination FROM flights a JOIN airlines b ON (a.airline = b.IATA_code) LIMIT 5; Time taken: 29.146 seconds, Fetched: 5 row(s)

Bucket Map Join

Bucket Map Join is used in cases where tables involved in Hive are large and tables are bucketed on same join key. Number of buckets in one table should be multiple of total number of buckets in another table. For example, if one t able has 3 buckets then other table should have buckets like 3,6,9 etc.

set hive.optimize.bucketmapjoin=true

Skew Join

This type of join works where you want to join tables in which one of the tables have one particular value in large n umber in join column. For example one stock_id in stock table is repeated
and it makes around 40% of total values in that table. You have to set following two properties in hive to enable ske w join.
hive.optimize.skewjoin=true;
set hive.skewjoin.key=100000

Sort Merge Bucket Join or SMB Join

When both tables are bucketed and sorted on join column and big tables has buckets in multiple of buckets from sma ll table.
and following properties are set to true
set hive.auto.convert.sortmerge.join=true
set hive.optimize.bucketmapjoin=true
set hive.optimize.bucketmapjoin.sortedmerge=true
set hive.auto.convert.sortedmerge.join.nonconditionaltask=true

UDF in Hive

Steps:
1) Create UDF
2) Package UDF
3) Adding UDF in Hive
4) Use UDF

Code to create a simple udf for getting square of a number

```
package com.demo.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;

public class square extends  UDF {
   public long evaluate(long number){
      return number*number;
   }
}
```

Package code in a jar file

Add jar file in hive
---

hive> add jar /mnt/home/rishabhtiwari2048gmail/Hive/square.jar

Create function using jar file in hive

Note: Always name your UDF function in lowercase letters
hive> create function square as 'com.demo.hive.udf.square'; --string is quotes is classpath name

Use function in your queries
---

hive> select num, square(num) from tablnumber;

Two types of UDFs
1) Regular UDF (User defined functions)
2) User defined aggregate functions (UDAFs)

Hive Collections (Complex datatypes in Hive)

If the data that you want to work with is little bit on unstructured side, you can use hive collections.

1) Array
---

if data is like below we can use array data type

1,Rishabh,3000,ABC,Mumbai
2,Suyas,5000,DEF,Delhi
3,Shamanth,9000,XYZ,Banglore
4,Rekha,10000,STU,Chicago
5,Raman,12000,RST,Seattle
6,Shivani,8500,MNO$P,Dallas

0: jdbc:hive2://10.0.21.22:10000> create table array_demo (id int, name string, salary int, divisions array<string>, lo
cation string) row format delimited fields terminated by ',' collection items terminated by '$' lines terminated by '\n' s
tored as textfile;

hive> load data local inpath '/mnt/home/rishabhtiwari2048gmail/Hive/array.txt' into table array_demo;

hive> select * from array_demo;
OK
1 Rishabh 3000 ["A","B","C"] Mumbai
2 Suyas 5000 ["D","E","F"] Delhi
3 Shamanth 9000 ["X","Y","Z"] Banglore
4 Rekha 10000 ["S","T","U"] Chicago
5 Raman 12000 ["R","S","T"] Seattle
6 Shivani 8500 ["M","N","O","P"] Dallas
Time taken: 0.361 seconds, Fetched: 6 row(s)

hive> select name, division[0], division[2] from array_demo;

2) Map

If data in one column can be represented in form of Key Value pair. We can use map data structure in hive table. for example take the following data

```
1,Rishabh,3000,A$B$C,ppf#200$epf#500$nps#900$,Mumbai
2,Suyas,5000,D$E$F,ppf#200$epf#250$nps#900$,Delhi
3,Shamanth,9000,X$Y$Z,ppf#200$epf#500$,Banglore
4,Rekha,10000,S$T$U,ppf#200$nps#900$,Chicago
5,Raman,12000,R$S$T,epf#500$nps#900$,Seattle
6,Shivani,8500,M$N$O$P,ppf#200$epf#500$,Dallas
```

```
hive (rishabhdb)> create table map_demo(id int, name string, salary int, divisions array<string>, funds map<string, i
nt>, city string) row format delimited fields terminated by ',' collection items terminated by '$' map keys terminated
by '#' lines terminated by '\n' stored as textfile;
```

```
hive (rishabhdb)> load data local inpath '/mnt/home/rishabhtiwari2048gmail/Hive/map.txt' into table map_demo;
```

```
hive (rishabhdb)> select * from map_demo;
OK
1       Rishabh 3000    ["A","B","C"]   {"ppf":200,"epf":500,"nps":900,"":null} Mumbai
2       Suyas   5000    ["D","E","F"]   {"ppf":200,"epf":250,"nps":900,"":null} Delhi
3       Shamanth        9000    ["X","Y","Z"]   {"ppf":200,"epf":500,"":null}    Banglore
4       Rekha   10000   ["S","T","U"]   {"ppf":200,"nps":900,"":null}    Chicago
5       Raman   12000   ["R","S","T"]   {"epf":500,"nps":900,"":null}    Seattle
6       Shivani 8500    ["M","N","O","P"]       {"ppf":200,"epf":500,"":null}    Dallas
Time taken: 0.076 seconds, Fetched: 6 row(s)
```

```
hive (rishabhdb)> select name, funds["ppf"], funds["nps"] from map_demo;

Query ID = rishabhtiwari2048gmail_20210417074342_dadfada8-bded-4ed4-a96b-f68df8f525a5
Total 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_1608530820093_31242, Tracking URL = http://ip-10-0-21-131.ec2.internal:8088/proxy/applicati
on_1608530820093_31242/
Kill Command = /opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/lib/hadoop/bin/hadoop job  -kill job_1608
530820093_31242
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-04-17 07:43:48,278 Stage-1 map = 0%,  reduce = 0%
2021-04-17 07:43:54,392 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.53 sec
MapReduce Total cumulative CPU time: 2 seconds 530 msec
Ended Job = job_1608530820093_31242
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 2.53 sec   HDFS Read: 5267 HDFS Write: 247 HDFS EC Read: 0 SUCC
ESS
Total MapReduce CPU Time Spent: 2 seconds 530 msec
OK
Rishabh 200     900
Suyas   200     900
Shamanth        200     NULL
Rekha   200     900
Raman   NULL    900
Shivani 200     NULL
Time taken: 13.26 seconds, Fetched: 6 row(s)
```

3) Struct

Struct data structure can be used in scenarios where you have a mix of primitive data types like below and you want to handle each value of the data as a column you can use struct

1,Rishabh,3000,ABC,MumbaiMAH400101
2,Suyas,5000,DEF,DelhiDEL110001
3,Shamanth,9000,XYZ,BangloreKAR560047
4,Rekha,10000,STU,ChicagoILL60604
5,Raman,12000,RST,SeattleWAS98101
6,Shivani,8500,MNO$P,Dallas$TEX$75204

hive (rishabhdb)> create table struct_demo(id int, name string, salary int, divisions array<string>, funds map<string, int>, address struct<city:string,state:string,pincode:bigint>) row format delimited fields terminated by ',' collection it ems terminated by '$' map keys terminated by '#' lines terminated by '\n' stored as textfile;

hive (rishabhdb)> load data local inpath '/mnt/home/rishabhtiwari2048gmail/Hive/struct.txt' into table struct_demo;

hive (rishabhdb)> select * from struct_demo;
OK
1 Rishabh 3000 ["A","B","C"] {"city":"Mumbai","state":"MAH","pincode":400101}
2 Suyas 5000 ["D","E","F"] {"city":"Delhi","state":"DEL","pincode":110001}
3 Shamanth 9000 ["X","Y","Z"] {"city":"Banglore","state":"KAR","pincode":560047}
4 Rekha 10000 ["S","T","U"] {"city":"Chicago","state":"ILL","pincode":60604}
5 Raman 12000 ["R","S","T"] {"city":"Seattle","state":"WAS","pincode":98101}
6 Shivani 8500 ["M","N","O","P"] {"city":"Dallas","state":"TEX","pincode":75204}
Time taken: 0.073 seconds, Fetched: 6 row(s)

hive (rishabhdb)> select name, address.city, address.state, address.pincode FROM struct_demo;
Query ID = rishabhtiwari2048gmail_20210417081220_e8bb3d4a-6f98-433c-9b6c-dcbaf594db5b
Total 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_1608530820093_31248, Tracking URL = http://ip-10-0-21-131.ec2.internal:8088/proxy/applicati on_1608530820093_31248/
Kill Command = /opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/lib/hadoop/bin/hadoop job -kill job_1608 530820093_31248
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-04-17 08:12:46,259 Stage-1 map = 0%, reduce = 0%
2021-04-17 08:12:52,378 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.44 sec
MapReduce Total cumulative CPU time: 1 seconds 440 msec
Ended Job = job_1608530820093_31248
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 1.44 sec HDFS Read: 5053 HDFS Write: 247 HDFS EC Read: 0 SUCC ESS
Total MapReduce CPU Time Spent: 1 seconds 440 msec
OK
Rishabh Mumbai
Suyas Delhi
Shamanth Banglore
Rekha Chicago
Raman Seattle
Shivani Dallas
Time taken: 33.201 seconds, Fetched: 6 row(s)