DADS 6002 / CI 7301
Big Data Analytics

Hive

- Hive is a data warehousing framework built on top of Hadoop

- It provides data analysts with a familiar SQL-based interface to Hadoop and so allow structure schema to be attached to data in HDFS.

- Hive Query language (HQL) support many commonly used SQL statements including

  Data Definition Statement (DDL) e.g. CREATE Database/Schema/Table, Data Manipulation Statement (DML) e.g. INSERT, UPDATE, LOAD, and Data Retrieval e.g. SELECT

Hive

- It also supports integration of custom user-defined functions written in JAVA or other languages supported by Hadoop Streaming.
- Hive commands and HQL queries are compiled into an execution plan or a series of HDFS operations and/or MapReduce jobs.
- Hive has inherited limitation of HDFS WORM so it does not provide in place data update.
- It does provide inserts, Update or Delete (ACID operations) but they are not very efficient to perform these operation frequently.

Hive

- Changes resulting from inserts, updates, and deletes are stored in small delta files.
- Delta files are periodically merged into base table files by MapReduce jobs that are run in the background (by the Metastore server).
- A Hive table can also be indexed to speed up queries to access the table.

Hive

- Hive maintains its database meta data in a separated MetaStore usually in MySQL database server (not in HDFS) to efficiently update and retrieve meta data.

- Hive provides scalable, high throughput batch-level data analytic processing for very large datasets.

Hive Command Line Interface (CLI)

- To start Hive CLI

  # hive

  hive >

- To exit Hive CLI

  hive > exit;

Hive

- Execute inline Hive command without entering to interactive mode.

  # hive –e ‘show databases’

- Execute the sequence of Hive commands in the hql file.

  # hive –f exec.hql

- Display the usage of the hive command

  # hive –h

HQL

- Create a database

  hive > create database log_data;

  Hive creates a new database which schema definition is stored in the Hive Metastore. If the database is already exist, Hive will raise error.

  hive > create database if not exists log_data;

  (create database if not exists)

HQL

- Show names of all existing databases

  hive > show databases;

- Set a working database

  hive > use log_data;

- Show tables in the working database

  hive > show tables;

- Show the schema of a table

  hive > describe test;

- Add a field in the schema of a table

  hive > alter table test add columns (address string);

- Delete a table

  hive > drop table test;

Hive

Create a table

- By default, when we create a table in Hive, Hive will manage the data, which means that Hive moves the data into its warehouse directory.
- Alternatively, we may create an external table, which tells Hive to refer to the data that is at an existing location outside the warehouse directory.

Hive

- When we delete (drop) an external table, Hive will leave the data untouched and only delete metadata.
- For a managed table

Hive > create table memo (lineno string, linetext string) row format delimited fields terminated by ‘\t’ stored as textfile;

Row format is used to indicate that the data file (in HDFS) has \t character as delimiter between fields.

Note : A table can be stored as ORC (optimized columnar format)

- For an external table

Hive > create external table memo (lineno string, linetext string) row format delimited fields terminated by ‘\t’ stored as textfile location ‘/user/cloudera/external_table’;

HQL

- Hive provides a way to apply a regex (regular expression) to known record formats to deserialize or parse each row into its constituent fields.

```
hive > create table apache_log (
                host  string,
                identity  string,
                user string,
                time  string,
```

HQL

```
        request  string,
        status string,
        size  string,
        referer  string,
        agent  string )
row format serde
‘org.apache.hadoop.hive.serde2.RegSerde’ with
Serdeproperties (“input.regex” =“([^ ]*) ([^ ]*) ([^ ]*) (-
|\\[[^\\]]*\\]) ([^ \”]*\”) (-|[0-9]*) (-|[0-9]*) (?: ([^ \”]*|\”.*\”) ([^
\”]*|\”.*\”))?”, ‘output.format.string’ = “%1$s %2$s %3$s
%4$s %5$s %6$s %7$s %8$s %9$s”) stored as textfile;
```

HQL

- Serde (Serializer/Deserializer) along with regular expression can be used to specify record format of the input file, which may not be in Hive format, so the content of the input file can be loaded into a Hive table.
- Regular Expressions can be extremely complex but they are very flexible and powerful and can be used to perform comparisons of input strings.

HQL

- ‘^' and '$'

  These symbols indicate the start and the end of a string, respectively: examples,
- "^The" - matches a string that starts with "The".
- "of despair$” - matches a string that ends in with "of despair".
- "^abc$" - a string that starts and ends with "abc" - effectively an exact match comparison.
- “[^]” - ^ in a bracket indicates not a following character e.g. not a space character.

HQL

- '*', '+', and '?'

  the symbols '*', '+', and '?', denote the number of times a character or a sequence of characters may occur. What they mean is: "zero or more", "one or more", and "zero or one." , examples:
- "ab*c" - matches a string that has an a followed by zero or more b's ("ac", "abc", "abbc", etc.)
- "ab+c" - same, but there's at least one b ("abc", "abbc", etc., but not "ac")
- "ab?c" - there might be a single b or not ("ac", "abc" but not "abbc").
- "a?b+$" - a possible 'a' followed by one or more 'b's at the end of the string: Matches any string ending with "ab", "abb", "abbb" etc. or "b", "bb" etc. but not "aab", "aabb" etc.

HQL

- Braces { } – bounds can appear inside braces and indicate ranges in the number of occurrences: examples
- "ab{2}“ - matches a string that has an a followed by exactly two b's ("abb")
- "ab{3,5}“ - from three to five b's (e.g. "abbb", "abbbb", or "abbbbb")
- ( ) - to quantify a sequence of characters, put them inside parentheses: examples,
- "a(bc)*“ - matches a string that has an a followed by zero or more copies of the sequence "bc"

HQL

- '|' the symbol, which works as an OR operator, examples,
- "hi|hello“ - matches a string that has either "hi" or "hello" in it
- "(b|cd)ef“ - a string that has either "bef" or "cdef“
- "(a|b)*c“ - a string that has a sequence of alternating a's and b's ending in a c

HQL

- [ ] Bracket expressions - specify which characters are allowed in a single position of a string:
- "[ab]" - matches a string that has either an a or a b (that's the same as "a|b")
- "[a-d]" - a string that has lowercase letters 'a' through 'd' (that's equal to "a|b|c|d" and even "[abcd]")
- "^[a-zA-Z]" - a string that starts with a letter
- "[0-9]%" - a string that has a single digit before a percent sign

HQL

- ('.') - a period ('.') stands for any single character: examples,
- "a.[0-9]" - matches a string that has an a followed by one character and a digit.
- "^.{3}$“ - a string with exactly 3 characters.
- “\” - an escape character used to match a character having special meaning in regex e.g. “\”” matches a double quote character.

Primitive Data Types

• tinyint	8 bit integer (-128 to 127)
• smallint	16 bit integer
• int	32 bit integer
• bint	64 bit integer
• float	32 bit single precision float
• double	64 bit double precision float
• boolean	true/false
• string	2 GB max character string
• timestamp	nanosecond precision (yyyy-mm-dd hh:mm:ss.fffffffff)

Complex Data Types

- array ordered collection of elements e.g.
  array<string>
- map unordered collection of key/value pairs, key must be primitive data type e.g.
  map<string,int>
- struct collection of elements of any types e.g.
  struct<name:string, age:int>

Hive

- By default, when we create a table in Hive, Hive will manage the data, which means that Hive moves the data into its warehouse directory.
- Alternatively, we may create an external table, which tells Hive to refer to the data that is at an existing location outside the warehouse directory.
- When we drop an external table, Hive will leave the data untouched and only delete metadata.

Loading Data

- When loading data into a table, Hive does not perform any verification of the data for compliance with the table schema.
- Hive can only enforce queries on schema read.
- If in reading a data file, the file structure does not match the defined schema, Hive returns Null values for missing fields or type mismatches.

Loading Data

- Data loading in Hive is done in batch mode using a bulk load data command or by inserting results from another query into a table with insert command.

  hive > use log_data;

  hive > load data inpath ‘/user/cloudera/apache.log’ overwrite into table apache_log;

Loading Data

hive > select count(1) from apache_log;

(A MapReduce job is started to perform the aggregation count)

- Insert results from select command into a table

hive > insert overwrite table apache_log

select * from source;

Data Analysis with Hive

Grouping

Consider a MapReduce program that computes the group-count problem. It requires a decent level of effort to write mapper, reducer and main function to configure the MapReduce job. With Hive, this problem can be done using a SQL group by query.

```
hive > select month, count(1) as count from
        (select split(time, '/')[1] as month from
        apache_log )
        group by month order by count desc;
```

Data Analysis with Hive

hive > select host, count(1) as count from apache_log group by host order by count;

- In addition to count Hive supports other aggregate functions e.g. sum, average, min, max, variance, standard deviation and covariance of numeric column (field).

hive > set hive.map.aggr = true;

- This setting tells Hive to perform “top level” partial aggregation in the map phase (combiner), as opposed to aggregation after performing group by. However, it would require more memory to compute the map phase.

Grouping

Hive allows creating a new table to store the results returned by other queries for later record-keeping and analysis.

```
hive > create table  remote_hits_by_month
        as select month, count(1) as count
        from ( select split(time, '/')[1] as month
        from apache_log where host == 'remote' )
        group by month  order by count desc;
```

Joins

- Two tables can be joined using inner join, outer join or semi join operations.
- Given two tables sales and things
- Table sales consists of name of customer (cname) and the IDs of the items they bought (id), e.g. sales.cname and sales.id.
- Table things consists of IDs of items (id) and their names (iname), e.g. things.id and things.iname.

Join

```
Hive > select * from sales;
Joe     2
Hank    4
Ali     0
Eve     3
Hank    2
Hive > select * from things;
2       Tie
4       Coat
3       Hat
1       Scarf
```

- ;

Inner Join

```
hive > select sales.*, things.* from sales join things on
       (sales.id = things.id );
```

Joe	2	2	Tie
Hank	4	4	Coat
Eve	3	3	Hat
Hank	2	2	Tie

Outer Join

hive > select sales.*, things.* from sales left outer join things on (sales.id = things.id);

Joe	2	2	Tie
Hank	4	4	Coat
Ali	0	NULL	NULL
Eve	3	3	Hat
Hank	2	2	Tie

Outer Join

```
hive > select sales.*, things.* from sales right outer
join things on (sales.id = things.id);
Joe             2               2       Tie
Hank            2               2       Tie
Hank            4               4       Coat
Eve             3               3       Hat
NULL            NULL            1       Scarf
```

Outer Join

```
hive > select sales.*, things.* from sales full outer
join things on (sales.id = things.id);
```

Ali	0	NULL	NULL
NULL	NULL	1	Scarf
Hank	2	2	Tie
Joe	2	2	Tie
Eve	3	3	Hat
hank	4	4	Coat

Semi Join

```
hive > select * from things left semi join sales on ( sales.id =
       things.id);
```

2 Tie

4 Coat

3 Hat

Types of Join Operations

1. **Left** Join
2. **Right** Join
3. **Full** Join
4. **Inner** Join
5. **Semi** Join
6. **Anti** Join