EDU PRISTINE

Apache Hive

Data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage and queried using SQL syntax.

© **EduPristine** – www.edupristine.com

Hive Features ?

Hive is build at the top of Map Reduce and provides below features

- Tools to enable easy access to data via SQL, thus enabling data warehousing tasks such as extract/transform/load (ETL), reporting, and data analysis.
- A mechanism to impose structure on a variety of data formats
- Query execution via various executions engines like Tez, Spark and Map Reduce
- Provides standard SQL functionality, including many of the later SQL:2003 and SQL:2011 features for analytics.
- There is no specific Hive format that data must be stored, but various connectors to read and write Text, CSV data. More connectors can be added !

What Hive Can do ?

- Hive Data Definition Language
  - CREATE/ DROP /ALTER / USE Database
  - CREATE/ DROP/Truncate Table
  - ALTER Table/Partition/Column
  - CREATE/ALTER/DROP View
  - CREATE/ALTER/DROP Index
  - CREATE/DROP/GRANT/REVOKE Roles and Privileges
- Hive Data Manipulation Language
  - LOAD, INSERT, UPDATE, DELETE
- Data Retrieval
  - SELECT, WHERE, SORT
  - GROUP BY
  - JOINS
  - UNIONS
  - SUB QUERIES
  - EXPLAIN
- File Formats
  - Avro File Format
  - ORC File Format
  - Parquet
  - LZO Compression

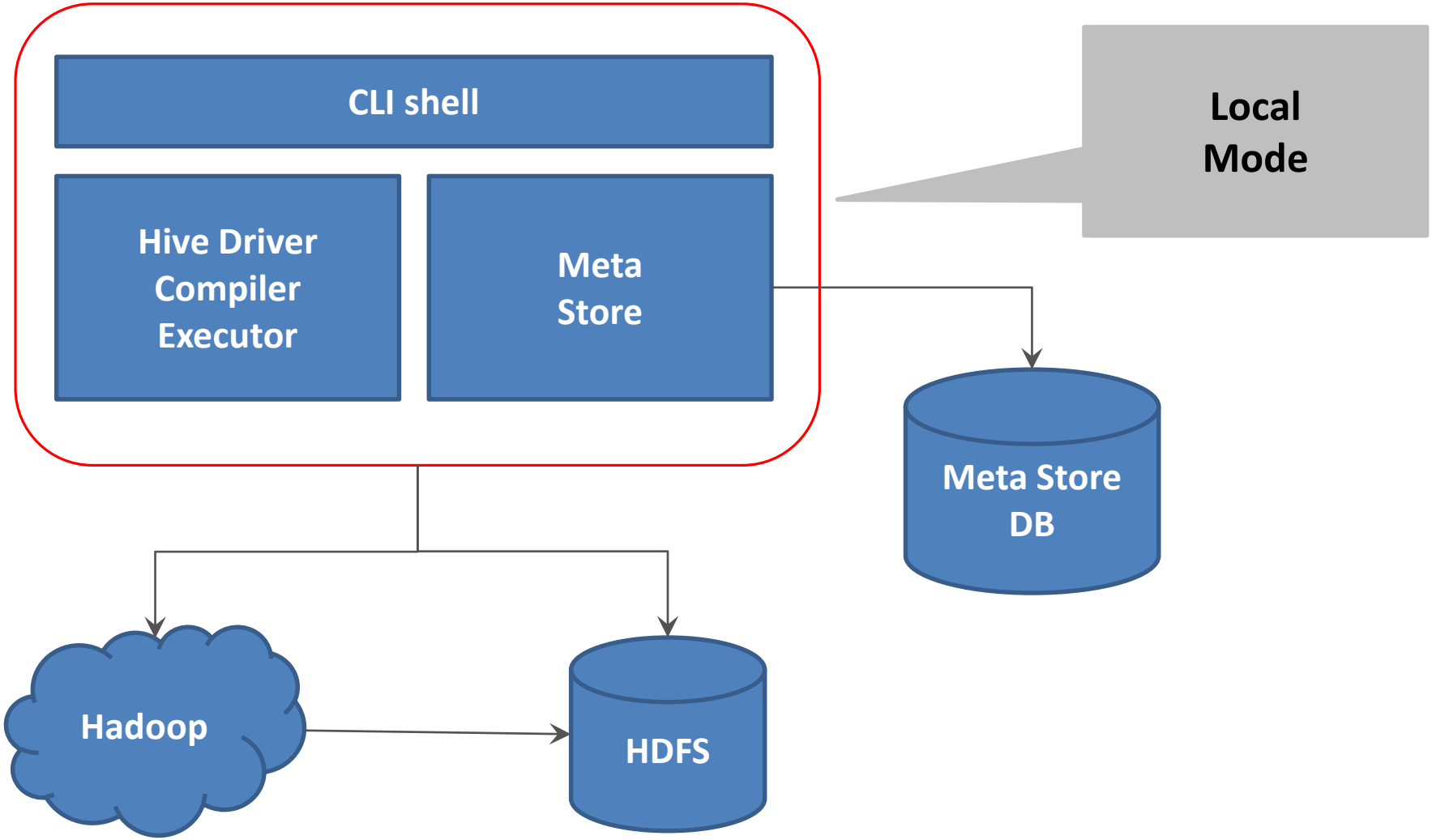
What HIVE is not ?

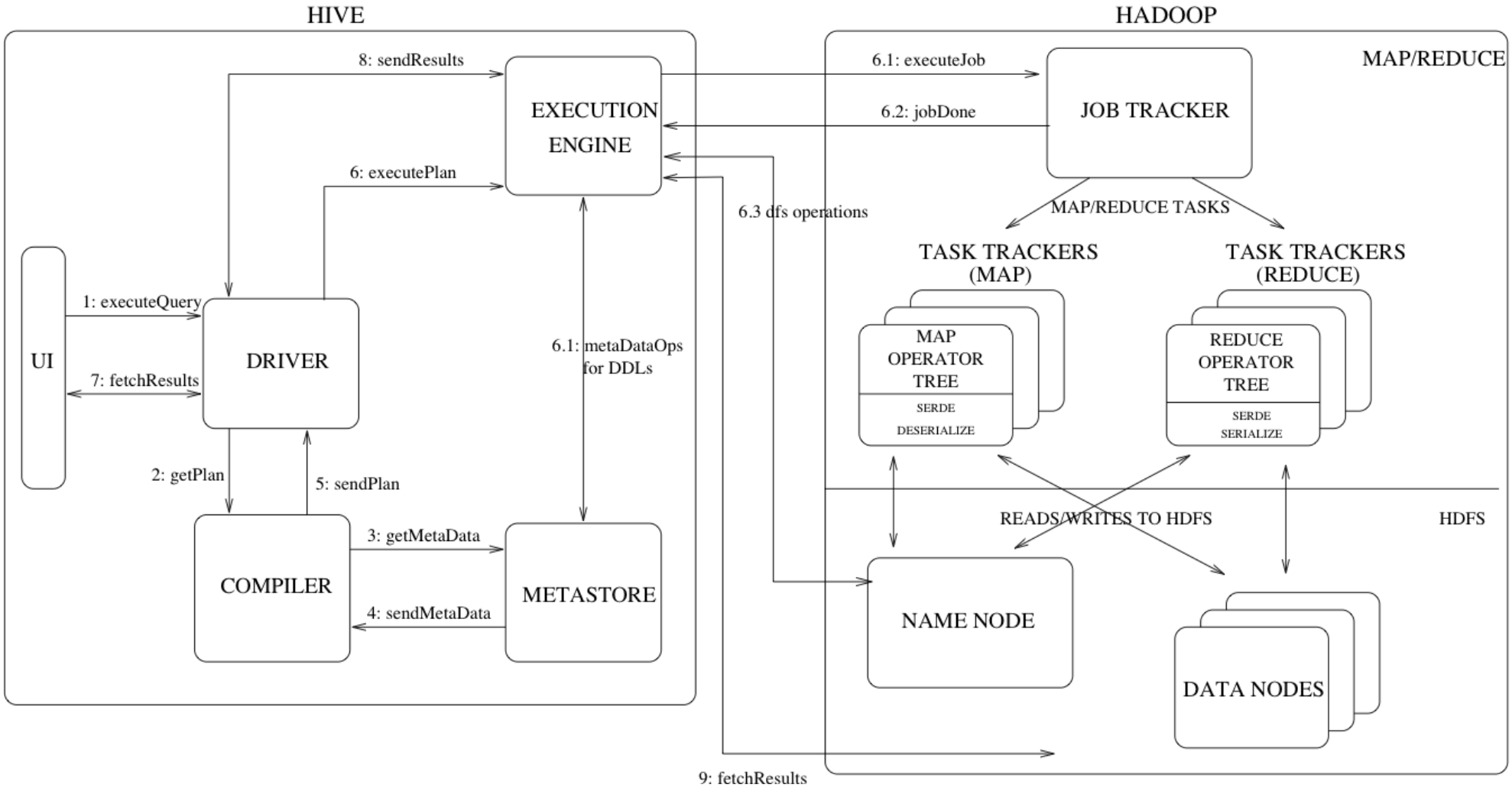
- Hive is not a RDBMS !
- Hive is not designed for online transaction processing (OLTP) workloads.
- It is best used for traditional data warehousing tasks.
- Correlated subqueries
- Even with small amount of data time to return the response can’t be compared to RDBMs

Connecting to HIVE

- Hive shell
- JDBC Driver
- ODBC Driver
- Thrift Client

Hive Architecture




HIVE
HADOOP
MAP/REDUCE
8: sendResults
6.1: executeJob
6.2: jobDone
EXECUTION ENGINE
JOB TRACKER
6: executePlan
6.3 dfs operations
MAP/REDUCE TASKS
TASK TRACKERS (MAP)
TASK TRACKERS (REDUCE)
1: executeQuery
UI
DRIVER
7: fetchResults
6.1: metaDataOps for DDLs
MAP OPERATOR TREE
SERDE DESERIALIZE
REDUCE OPERATOR TREE
SERDE SERIALIZE
2: getPlan
5: sendPlan
READS/WRITES TO HDFS
HDFS
3: getMetaData
COMPILER
METASTORE
4: sendMetaData
NAME NODE
DATA NODES
9: fetchResults

Hive Components

- HCatalog - is a component of Hive. It is a table and storage management layer for Hadoop that enables users with different data processing tools — including Pig and MapReduce — to more easily read and write data on the grid.
- WebHCat - provides a service that you can use to run Hadoop MapReduce (or YARN), Pig, Hive jobs or perform Hive metadata operations using an HTTP (REST style) interface

Hive MetaStore

- Persists Schema, i.e. table definition(table name, columns, types)
- Location of table files
- Row format of table files
- Storage format of files

- Exercise
  - Check Hive configuration to know DB instance [MySQL in this case] and of the meta-store DB in
  - Log in to MySQL Instance on your cluster and look for Hive meta-store DB.

Lets do Hive !

- We are going to use data of truck driver statistics for some simple computing using Hive.

- Use ‘driver_dataset’ provided for this session
  - **drivers.csv** is comma separated file with fields as *driverId, name, ssn, location, certified, wage-plan*
  - **Timesheet.csv** is comma separated file with fields as *driverId, week, hours-logged, miles-logged*
  - **Truck_events_partition.csv** is comma separated file with fields as *driverId, truckId, eventTime, eventType, longitude, latitude, eventKey, CorrelationId, driverName, routeId, routeName, eventDate*

1 Create and use a new database in hive

```
CREATE DATABASE IF NOT EXISTS MYDB;
USE MYDB;
```

2 Create a managed table stored as Text File use a new database in hive

```
CREATE TABLE MYDB.drivers (
driverId INT,
name STRING,
ssn BIGINT,
location STRING,
certified STRING,
wageplan STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Learn & Discuss

- Default database & working with multiple databases
- Managed & External Tables
- ROW Format, Field Separator
- File Formats

Note: verify directory structure changes in Hive warehouse in HDFS to understand managed table's location

Option 1: Load from Local File System

LOAD DATA LOCAL INPATH '{your local path to}/drivers.csv' INTO TABLE MYDB.drivers;

Option 2: Load from HDFS File System

LOAD DATA INPATH '{your HDFS path to}/drivers.csv' INTO TABLE MYDB.drivers;

Verify Data.

select count(*) from MYDB.drivers;

Learn & Discuss

- Load data from local and HDFS file to Hive table

Create timesheet table and populate data. Overwrite table while loading

```
CREATE TABLE MYDB.timesheet (
 driverId INT,
 week INT,
 hours_logged INT,
 miles_logged INT
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ',';
```

Learn & Discuss

- Overwrite table while loading data from local and HDFS file to Hive table

Option 1: Load from Local File System

```
LOAD DATA LOCAL INPATH '{your local path to}/timesheet.csv'  OVERWRITE INTO TABLE MYDB.timesheet;
```

Option 2: Load from HDFS File System

```
LOAD DATA INPATH '{your HDFS path to}/timesheet.csv'  OVERWRITE INTO TABLE MYDB.timesheet;
```

Verify table and data

```
describe MYDB.timesheet;
select count(*) from MYDB.timesheet;
```

Find out per driver total miles and hours driven

```
SELECT driverId, sum(hours_logged), sum(miles_logged) FROM MYDB.timesheet GROUP BY driverId;
```

Join with drivers table, to get drivers details

```
SELECT d.driverId, d.name, t.total_hours, t.total_miles from MYDB.drivers d
JOIN (SELECT driverId, sum(hours_logged)total_hours, sum(miles_logged)total_miles FROM MYDB.timesheet
GROUP BY driverId ) t
ON (d.driverId = t.driverId);
```

- Hive supports
  - Inner joins
  - Left outer joins
  - Right outer joins
  - Full outer joins

Find out per driver total miles and hours driven.
Populate Data into Table in ORC Format

Create ORC table to store result of above query.

CREATE TABLE MYDB.driver_aggregate_orc (

driver_id INT,

driver_name STRING,

total_hours INT,

total_miles INT

)

STORED AS ORC;

Learn
- Create table with ORC format and load data
- ORC table can not be directly loaded using LOAD

Populate Data

INSERT INTO MYDB.driver_aggregate_orc SELECT d.driverId, d.name, t.total_hours, t.total_miles from MYDB.drivers d

JOIN (SELECT driverId, sum(hours_logged)total_hours, sum(miles_logged)total_miles FROM MYDB.timesheet GROUP BY driverId) t

ON (d.driverId = t.driverId);

Create Partitioned Table for truck_events_data. Use eventDate as partition key. Table should be stored as Sequence File.

```
CREATE TABLE truck_events_partitioned (
 driverId INT,
 truckId INT,
 eventTime STRING,
 eventType STRING,
 longitude STRING,
 latitude STRING,
 eventKey STRING,
 correlationId STRING,
 driverName STRING,
 routeId STRING,
 routeName STRING,
 eventDate STRING
)
 PARTITIONED BY(eventDateKey STRING)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
 STORED AS SEQUENCEFILE;
```

Learn & Discuss

- Partitioned Table
- Partitioned Key name can not match with any column
- Populate data in appropriate partition

Try some simple analysis

Learn & Discuss

- Order By, Limit

Who drove maximum miles ?

```
select * from MYDB.driver_aggregate_orc order BY total_miles DESC LIMIT 1;
```

Least ?

```
select * from MYDB.driver_aggregate_orc order BY total_miles LIMIT 1;
```

Create External Table for truck_events_data and load data

```
CREATE EXTERNAL TABLE truck_events_external (
 driverId INT,
 truckId INT,
 eventTime STRING,
 eventType STRING,
 longitude STRING,
 latitude STRING,
 eventKey STRING,
 correlationId STRING,
 driverName STRING,
 routeId STRING,
 routeName STRING,
 eventDate STRING
)
 LOCATION '/user/{your user}/truck_ext'
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
 STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '{your local path to}/truck_event.csv' INTO TABLE truck_events_external ;
```

Learn & Discuss

- External Table
- Partitioned Key name can not match with any column
- Populate data in appropriate partition

Load data into Partitioned table. Data should go to appropriate partitions

INSERT INTO truck_events_partitioned PARTITION(eventDateKey) SELECT driverId, truckId, eventTime, eventType, longitude, latitude, eventKey, CorrelationId, driverName, routeId, routeName, eventDate, eventDate FROM truck_events_external;

Got an Error ? Set dynamic partition to nonstrict

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

Try Now again !

Verify partitons

DESCRIBE truck_events_partitioned;

SHOW PARTITIONS truck_events_partitioned;

Learn & Discuss

- Use of External Table to populate partitioned table
- Load using dynamic partitioning

Create Index on Existing Table

INSERT INTO truck_events_partitioned PARTITION(eventDateKey) SELECT driverId, truckId, eventTime, eventType, longitude, latitude, eventKey, CorrelationId, driverName, routeId, routeName, eventDate, eventDate FROM truck_events_external;

Got an Error ? Set dynamic partition to nonstrict

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

Try Now again !

Verify partitons

DESCRIBE truck_events_partitioned;

SHOW PARTITIONS truck_events_partitioned;

Learn & Discuss

- Use of External Table to populate partitioned table
- Load using dynamic partitioning

Configuring hive on vm(hive-site.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- Hive Configuration can either be stored in this file or in the hadoop configuration files  -->
<!-- that are implied by Hadoop setup variables.                                               -->
<!-- Aside from Hadoop setup variables - this file is provided as a convenience so that Hive    -->
<!-- users do not have to edit hadoop configuration files (that may be managed as a centralized -->
<!-- resource).                                                                                 -->

<!-- Hive Execution Parameters -->

<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby:;databaseName=/home/cloudera/Desktop/metastore/metastore_db;create=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>org.apache.derby.jdbc.EmbeddedDriver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>

<property>
  <name>hive.hwi.war.file</name>
  <value>/usr/lib/hive/lib/hive-hwi-0.7.0-cdh3u0.war</value>
  <description>This is the WAR file with the jsp content for Hive Web Interface</description>
</property>

</configuration>
~
~
```

HealthCheck

- After configuring hive, type command show tables; if you get output without any error, hive is configured properly.
- The logs of hive for debugging can be checked at /tmp/{username}/hive.log. If you did sudo hive with root the logs will be at /tmp/root/hive.log otherwise /tmp/cloudera/hive.log.

Data types

Numeric Types

TINYINT (1-byte signed integer, -128 to 127)

SMALLINT (2-byte signed integer, from -32,768 to 32,767)

INT/INTEGER (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)

BIGINT (8-byte signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

FLOAT (4-byte single precision floating point number)

DOUBLE (8-byte double precision floating point number)

DOUBLE PRECISION (alias for DOUBLE, only available starting with Hive 2.2.0)

DECIMAL

Date/Time Types

TIMESTAMP (Note: Only available starting with Hive 0.8.0)

DATE (Note: Only available starting with Hive 0.12.0)

INTERVAL (Note: Only available starting with Hive 1.2.0)

Misc. Types

BOOLEAN

BINARY (Note: Only available starting with Hive 0.8.0)

String Types

STRING

VARCHAR (Note: Only available starting with Hive 0.12.0)

CHAR (Note: Only available starting with Hive 0.13.0)

Complex Types

arrays: ARRAY<data_type>

(Note: negative values and non-constant expressions are allowed as of Hive 0.14.)

maps: MAP<primitive_type, data_type>

(Note: negative values and non-constant expressions are allowed as of Hive 0.14.)

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

union: UNIONTYPE<data_type, data_type, ...> (Note: Only available starting with Hive 0.7.0.)

Syntax of collections

- Create table employee (id int, name String, salary Double, skills Array<String>)
  - ROW FORMAT DELIMITED
  - FIELDS TERMINATED BY '\t'
  - COLLECTION ITEMS TERMINATED BY ','
  - STORED AS TEXTFILE

External VS Managed Table

Use EXTERNAL tables when:

- The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.
- Data needs to remain in the underlying location even after a DROP TABLE. This can apply if you are pointing multiple schemas (tables or views) at a single data set or if you are iterating through various possible schemas.
- You want to use a custom location such as ASV.
- Hive should not own data and control settings, dirs, etc., you have another program or process that will do those things.
- You are not creating table based on existing table (AS SELECT).

Use MANAGED tables when:

- The data is temporary.
- You want Hive to completely manage the lifecycle of the table and data.

When external table is dropped only table is dropped i.e metadata is updated but data remains as is. If a managed table is dropped it also deletes the data.

Export Hive Table

- INSERT OVERWRITE DIRECTORY
  - To write result in HDFS
  - LOCAL to write on local FS

Export data from Hive table to local file system

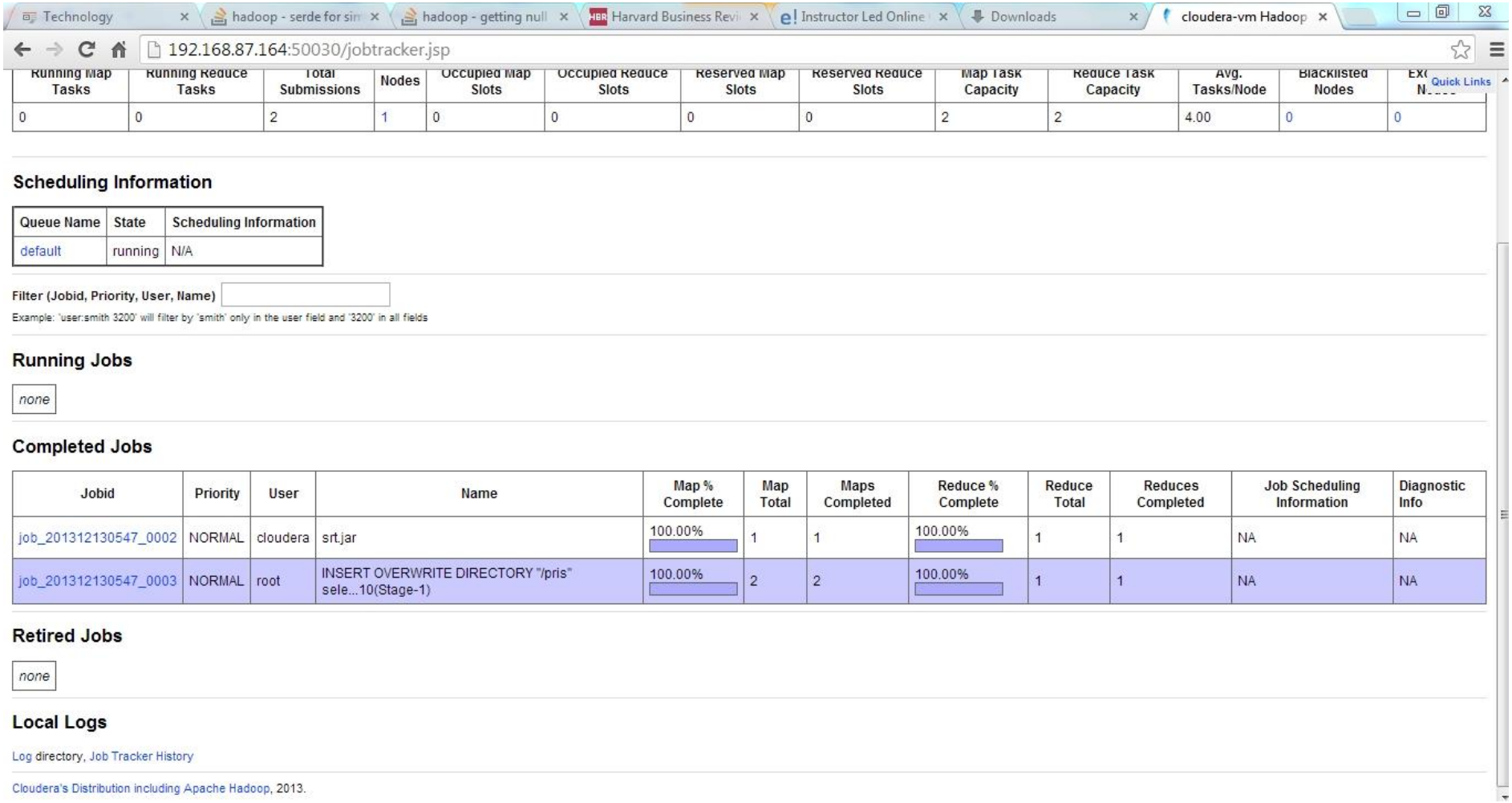
INSERT OVERWRITE LOCAL DIRECTORY ‘{path to local file}’
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' SELECT * FROM {your hive table name};

Export data from Hive table to HDFS

INSERT OVERWRITE DIRECTORY ‘{path to HDFS file}’
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' SELECT * FROM {your hive table name};

```
hive> INSERT OVERWRITE DIRECTORY "/pris" select * from SortedSubscribers LIMIT 10;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312130547_0003, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312130547_0003
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312130547_0003
2013-12-13 06:28:44,076 Stage-1 map = 0%,  reduce = 0%
2013-12-13 06:29:44,649 Stage-1 map = 0%,  reduce = 0%
2013-12-13 06:30:45,218 Stage-1 map = 0%,  reduce = 0%
2013-12-13 06:31:26,532 Stage-1 map = 79%,  reduce = 0%
2013-12-13 06:31:27,552 Stage-1 map = 100%,  reduce = 0%
2013-12-13 06:31:38,645 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312130547_0003
Moving data to: /pris
10 Rows loaded to /pris
OK
Time taken: 194.347 seconds
hive>
```

The progress of sql queries can also be checked at jobtracker page url:50030



Multiple DBs in Hive

- Default database is named default

```
root@cloudera-vm:/home/cloudera# sudo hive
Hive history file=/tmp/root/hive_job_log_root_201312130641_1913664056.txt
hive> show databases;
OK
default
Time taken: 21.884 seconds
hive> create database pristine;
OK
Time taken: 2.147 seconds
hive> use pristine;
OK
Time taken: 0.074 seconds
hive> show tables;
OK
Time taken: 0.868 seconds
hive> use default;
OK
Time taken: 0.032 seconds
hive> show tables;
OK
mytable
sortedsubscribers
sorts
subf
subsc
subscribers
subscribersunsorted
Time taken: 0.284 seconds
hive>
```

Oder by

- Select expr, expr .. from tablename

  where condition

  order by expr desc;

This will sort the data in descending order. The default ordering is ascending

Exercise

- Use order by clause to sort the subscribers output generated file by hive(from MR exercise)

Solution

```
hive> select id, bytes from mytable order by bytes;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312131243_0001, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0001
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0001
2013-12-13 12:44:45,148 Stage-1 map = 0%,  reduce = 0%
2013-12-13 12:45:45,757 Stage-1 map = 0%,  reduce = 0%
2013-12-13 12:46:43,001 Stage-1 map = 50%,  reduce = 0%
2013-12-13 12:46:44,012 Stage-1 map = 100%,  reduce = 0%
```

Sort by

- SORT BY expression
  - Sorts the data before going to reducer

Select expr, expr … from tablename

where condition

Sort by expr

Comparison Sort by and order by

- Sort by
  - May use multiple reducers for final output
  - Only guarantees ordering of rows within a reducer
  - May give partially ordered final result

- Order by
  - Uses single reducer to guarantee total order in output
  - LIMIT can be used to minimize sort time

Joining tables

- Hive supports
  - Inner joins
  - Left outer joins
  - Right outer joins
  - Full outer joins

Syntax

- Select cols FROM table1 join table2 on [condition]

Join on 2 tables

Employee

EMP ID	Emp Name	Address
1	Rose	US
2	Fred	US
3	Jess	In
4	Frey	Th

Employee Department

Emp ID	Department
1	IT
2	IT
3	Eng
4	Admin

Inner join

- Select * from employee join employee department ON(employee.empId=employeedepartment.empId)

EMP ID	Emp Name	Address	Emp ID	Department
1	Rose	US	1	IT
2	Fred	US	2	IT
3	Jess	In	3	Eng
4	Frey	Th	4	Admin

Left outer join

- Select e.empId, empName, department from employee e Left outer join employee department ed on (e.empId=ed.empId);

EMP ID	Emp Name	Department
1	Rose	IT
2	Fred	IT
3	Jess	Eng
4	Frey	Admin
5	Chris	null

Hive functions

- round()
- floor()
- ceil()
- rand()
- exp()
- length()
- concat()
- substr()
- lower()

  etc

Use of functions

```
hive> select sum(bytes) from subf;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312131243_0003, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0003
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0003
2013-12-13 14:02:05,006 Stage-1 map = 0%,  reduce = 0%
2013-12-13 14:02:32,477 Stage-1 map = 100%,  reduce = 0%
2013-12-13 14:02:45,700 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0003
OK
2.1219099097886E13
Time taken: 75.325 seconds
hive>
```

Views

- CREATE VIEW view(columnNames)

  As Select …

To remove view

Drop View view

View

```
hive> create view v(id) AS Select id from subf;
OK
Time taken: 0.998 seconds
hive> select * from v;
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_201312131243_0004, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0004
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0004
2013-12-13 14:05:55,347 Stage-1 map = 0%,  reduce = 0%
2013-12-13 14:05:58,499 Stage-1 map = 100%,  reduce = 0%
2013-12-13 14:06:01,528 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0004
OK
This is subscriberMR
11128052609
11128052610
11128052611
11128052612
11128052613
11128052614
11128052615
11128052616
11128052617
11128052618
11128052619
11128052620
11128052621
11128052622
11128052623
11128052624
11128052625
11128052626
11128052627
```

Partitioning

- Partitioning a data set means dividing and splitting the data into smaller partitions using values of columns.
- Hive partitions are stored in subdirectories of table directory

Creating custom UDFs

- To implement custom UDF, need to extend org.apache.hadoop.hive.ql.exec.UDF class present in hive-exec-0.7.0-cdh3u0. This jar can be copied from /usr/lib/hive/lib of VM.
- The evaluate method can be implemented accordingly

Custom UDF

ParseFunction.java

```
1  import org.apache.hadoop.hive.ql.exec.UDF;
2
3
4
5  public class ParseFunction extends UDF{
6      
7      public Double evaluate(Double input)
8      {
9          double bytes = input.doubleValue()/(1024*1024*1024);
10         return new Double(bytes);
11     }
12
13 }
14
```

How to run UDF

```
hive> add jar /home/cloudera/udf.jar;
Added /home/cloudera/udf.jar to class path
Added resource: /home/cloudera/udf.jar
hive> create temporary function toGB as 'ParseFunction';
OK
Time taken: 0.0080 seconds
hive> select toGB(bytes) from subf ;
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_201312131243_0012, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0012
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0012
2013-12-13 15:31:18,317 Stage-1 map = 0%,  reduce = 0%
2013-12-13 15:31:22,402 Stage-1 map = 100%,  reduce = 0%
2013-12-13 15:31:24,418 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0012
OK
7.450580596923828E-9
413.06346980668604
381.441385993734
371.5594848021865
389.346906946972
383.4177662320435
399.2288081385195
417.01623028330505
391.3232871852815
395.2760476619005
```

Serde

- Serde (Serializer/Deserializer) is used by Hive to control how lines are read and written to from files
- When used as serializer (i.e insert) table's serde will serialize hive's internal representation of row of data into bytes written to output file
- When used as deserializer (i.e querying the data), serde will deserialize a row of data from bytes in file to objects

Hive serde

SerDeName	Java Package	Description
LazySimpleSerDe	org.apache.hadoop.hive.serde2.lazy	The default serde. Delimited textual format
LazyBinarySerDe	org.apache.hadoop.hive.serde2.lazybinary	A more efficient version of LazySimpleSerde Binary format with lazy field access
BinarySortableSerDe	org.apache.hadoop.hive.serde2.binary sortable	Optimized for storing at expense of compactness
ColumnarSerDe	org.apache.hadoop.hive.serde2.columnar	For column based storage with RCFile
RegexSerDe	org.apache.hadoop.hive.contrib.serde2	For textual data where columns are specified by regular expression

Storage format

- 2 dimensions that control data storage in hive: Row Format and File Format
- Row format dictates how rows, and fields in a row, are stored
- Row format is defined by Serde (serializer/Deserializer)

Default storage format

- Default format is delimited text, with a row per line.
- Rows in a table are delimited by newline character.
- Hive uses LazySimpleSerde for this format

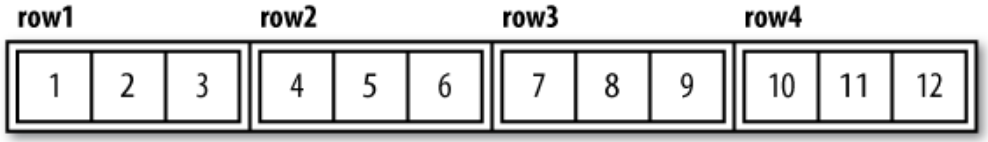
Binary storage format RCFile and sequencefile

- Using case study 1 of MR the output file generated by Subscriber Mapper and Reducer was sequence file.(key/value pairs)
- Sequence files are row oriented
- RCFiles, row columnar file. Similar to sequence file but data is stored in columnar fashion.

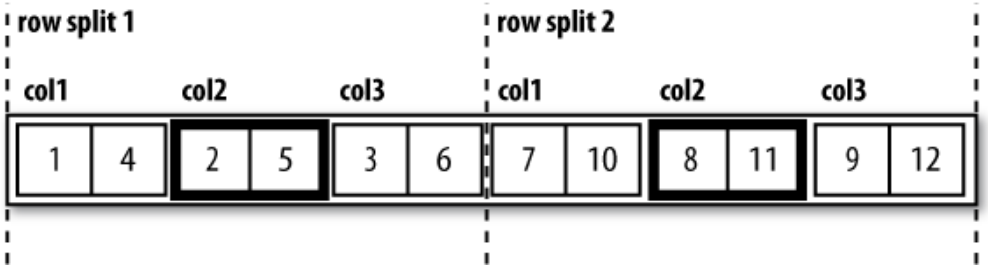
Logical table

	col1	col2	col3
row1	1	2	3
row2	4	5	6
row3	7	8	9
row4	10	11	12

Row-oriented layout (SequenceFile)



Column-oriented layout (RCFile)



Case study 2

- The output of the SubscriberMapper and SubscriberReducer was sequence file
- The create command should be Create external table SubscribersUnsorted(id String, bytes Double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS **SEQUENCEFILE** location '/output';

Sequence file

```
hive> Create external table SubscribersUnsorted(id String, bytes Double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS SEQUENCEFILE location '/output';
OK
Time taken: 0.107 seconds
hive> select * from SubscribersUnsorted;
OK
Failed with exception java.io.IOException:org.apache.hadoop.hive.serde2.SerDeException: class org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe: expects either BytesWr
itable or Text object!
Time taken: 0.194 seconds
hive>
```

```
14
15 public class PSequenceFileKeyInputFormat<K, V> extends FileInputFormat<K, V> {
16
17   public PSequenceFileKeyInputFormat() {
18     setMinSplitSize(SequenceFile.SYNC_INTERVAL);
19   }
20
21   @Override
22   protected FileStatus[] listStatus(JobConf job) throws IOException {
23     FileStatus[] files = super.listStatus(job);
24     for (int i = 0; i < files.length; i++) {
25       FileStatus file = files[i];
26       if (file.isDir()) {     // it's a MapFile
27         Path dataFile = new Path(file.getPath(), MapFile.DATA_FILE_NAME);
28         FileSystem fs = file.getPath().getFileSystem(job);
29         // use the data file
30         files[i] = fs.getFileStatus(dataFile);
31       }
32     }
33     return files;
34   }
35
36   public RecordReader<K, V> getRecordReader(InputSplit split,
37                                             JobConf job, Reporter reporter)
38     throws IOException {
39
40     reporter.setStatus(split.toString());
41
42     return (RecordReader<K, V>) new DoubleTextReader(job, (FileSplit) split);
43   }
44
45 }
46
```

ParseFunction.java | PristineSequenceFileInputFormat.java | PSequenceFileKeyInputFormat.java | DoubleTextReader.java

```
 1
 2
 3⊕import java.io.IOException;
10
11
12 public class DoubleTextReader extends PSequenceFileKeyRecordReader<Text, DoubleWritable>{
13
14⊖    public DoubleTextReader(Configuration conf, FileSplit split)
15             throws IOException {
16         super(conf, split);
17
18     }
19
20⊖    @Override
21     protected void combineKeyValue(Text key, DoubleWritable trueValue,
22             BytesWritable newValue) {
23             StringBuilder builder = new StringBuilder();
24             builder.append(key);
25             builder.append('\001');
26             builder.append(trueValue.get());
27             newValue.set(new BytesWritable(builder.toString().getBytes()) );
28
29     }
30
31⊖    @Override
32     public BytesWritable createValue() {
33         return new BytesWritable();
34     }
35
36 }
37
```

ParseFunction.java | PristineSequenceFileInputFormat.java | PSequenceFileKeyInputFormat.java | DoubleTextReader.java | PSequenceFileKeyRecordReader.java

```
1  import java.io.IOException;
11
12 public abstract class PSequenceFileKeyRecordReader<K, V> implements RecordReader<K, BytesWritable> {
13
14   private SequenceFile.Reader in;
15   private long start;
16   private long end;
17   private boolean more = true;
18   protected Configuration conf;
19
20   public PSequenceFileKeyRecordReader(Configuration conf, FileSplit split)
21     throws IOException {
22     Path path = split.getPath();
23     FileSystem fs = path.getFileSystem(conf);
24     this.in = new SequenceFile.Reader(fs, path, conf);
25     this.end = split.getStart() + split.getLength();
26     this.conf = conf;
27
28     if (split.getStart() > in.getPosition())
29       in.sync(split.getStart());                  // sync to start
30
31     this.start = in.getPosition();
32     more = start < end;
33   }
34
35
36   public Class getKeyClass() { return in.getKeyClass(); }
37
38
39   public Class getValueClass() { return in.getValueClass(); }
40
41
```

```
@SuppressWarnings("unchecked")
public K createKey() {
  return (K) ReflectionUtils.newInstance(getKeyClass(), conf);
}


public float getProgress() throws IOException {
  if (end == start) {
    return 0.0f;
  } else {
    return Math.min(1.0f, (in.getPosition() - start) / (float)(end - start));
  }
}

public synchronized long getPos() throws IOException {
  return in.getPosition();
}

protected synchronized void seek(long pos) throws IOException {
  in.seek(pos);
}
public synchronized void close() throws IOException { in.close(); }
```

```
65
66
67  @Override
68  public boolean next(K key, BytesWritable value) throws IOException {
69      if (!more) return false;
70
71      long pos = in.getPosition();
72      V trueValue = (V) ReflectionUtils.newInstance(in.getValueClass(), conf);
73      boolean remaining = in.next((Writable)key, (Writable)trueValue);
74      if (remaining) combineKeyValue(key, trueValue, value);
75      if (pos >= end && in.syncSeen()) {
76        more = false;
77      } else {
78        more = remaining;
79      }
80      return more;
81  }
82  protected abstract void combineKeyValue(K key, V trueValue, BytesWritable newValue);
83  }
```

```
root@cloudera-vm:/home/cloudera/Desktop/metastore/metastore_db# sudo hive
Hive history file=/tmp/root/hive_job_log_root_201312131900_120154923.txt
hive> add jar /home/cloudera/inp.jar;
Added /home/cloudera/inp.jar to class path
Added resource: /home/cloudera/inp.jar
hive> Create external table subscribersequencetable (id String, Bytes Double) STORED AS INPUTFORMAT 'PSequenceFileKeyInputFormat' OUTPUTFORMAT 'org.apache.hadoop.mapred
.SequenceFileOutputFormat' location '/output';
OK
Time taken: 24.265 seconds
hive> select * from subscribersequencetable;
OK
This is subscriberMR    8.0
11128052609     4.43523523498E11
11128052610     4.09569569546E11
11128052611     3.98958958936E11
11128052612     4.18058058034E11
11128052613     4.11691691668E11
11128052614     4.28668668644E11
11128052615     4.47767767742E11
11128052616     4.20180180156E11
11128052617     4.244244244E11
11128052618     4.0320320318E11
11128052619     3.86226226204E11
11128052620     4.05325325302E11
11128052621     4.1381381379E11
11128052622     4.30790790766E11
11128052623     3.56516516496E11
11128052624     4.26546546522E11
11128052625     3.84104104082E11
11128052626     3.94714714692E11
11128052627     3.9259259257E11
```

Exercise

- Put given csvInput file in exercise folder of hdfs. This file is sampleinput that gives the username and the site he accessed
- Write MR job to calculate which user visited how many sites
- Write MR job to calculate how many users visited the sites given in the file
- Write HIVE QL to calculate the above two counts
- Write hive QL to find out all the unique usernames in the table

```
hive> create external table urlcsvtable(name String, url String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE location '/exercise';
OK
Time taken: 25.877 seconds
hive> select name from urlcsvtable;
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_201312131243_0015, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0015
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0015
2013-12-13 20:33:26,460 Stage-1 map = 0%,  reduce = 0%
2013-12-13 20:33:35,879 Stage-1 map = 100%,  reduce = 0%
2013-12-13 20:33:37,899 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0015
OK
John
Mickey
Zhao
Zing
Zhao
Zhai
Zhao

Sam
John
jess
Chris
Cameron
Daisy
Chip
Dale
Sam
Time taken: 29.469 seconds
```

```
hive> select distinct name from urlcsvtable;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312131243_0016, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0016
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0016
2013-12-13 20:35:04,560 Stage-1 map = 0%,  reduce = 0%
2013-12-13 20:35:31,998 Stage-1 map = 50%,  reduce = 0%
2013-12-13 20:35:33,014 Stage-1 map = 100%,  reduce = 0%
2013-12-13 20:35:48,444 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0016
OK

Cameron
Chip
Chris
Daisy
Dale
John
Mickey
Sam
Zhai
Zhao
Zing
jess
Time taken: 51.526 seconds
```

```
hive> select name, count( distinct url) from urlcsvtable group by name;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312131243_0021, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0021
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0021
2013-12-13 20:47:57,496 Stage-1 map = 0%,  reduce = 0%
2013-12-13 20:48:11,686 Stage-1 map = 100%,  reduce = 0%
2013-12-13 20:48:23,756 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0021
OK
        0
Cameron 1
Chip    1
Chris   1
Daisy   1
Dale    1
John    2
Mickey  1
Sam     2
Zhai    1
Zhao    3
Zing    1
jess    1
Time taken: 33.597 seconds
```

```
hive> select url, count( distinct name) from urlcsvtable group by url;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312131243_0024, Tracking URL = http://cloudera-vm:50030/jobdetails.jsp?jobid=job_201312131243_0024
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:8021 -kill job_201312131243_0024
2013-12-13 20:58:53,803 Stage-1 map = 0%,  reduce = 0%
2013-12-13 20:59:00,885 Stage-1 map = 100%,  reduce = 0%
2013-12-13 20:59:11,987 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0024
OK
edupristine.com 8
facebook.com    5
google.com      3
Time taken: 37.479 seconds
```

```
hive> select url, count( distinct name) from urlcsvtable group by url;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201312131243_0024, Tracking URL = http://cloudera-vm:50030/jo
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=cloudera-vm:
2013-12-13 20:58:53,803 Stage-1 map = 0%,  reduce = 0%
2013-12-13 20:59:00,885 Stage-1 map = 100%,  reduce = 0%
2013-12-13 20:59:11,987 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312131243_0024
OK
edupristine.com 8
facebook.com    5
google.com      3
Time taken: 37.479 seconds
hive>
```

UrlMapper.java | SubscriberMapper.java | UrlReducer.java | SubscriberReducer.java | CountUsersMapper.java

```
1  import java.io.IOException;
6
7
8
9  public class UrlMapper  extends Mapper<LongWritable, Text, Text, Text>{
10
11     public void map(Text key, Text value, Context context)throws IOException, InterruptedException
12     {
13         String line = value.toString();
14         String[] nameAndUrl = line.split("\t");
15
16         Text nameKey = new Text(nameAndUrl[0]);
17         Text urlValue = new Text(nameAndUrl[1]);
18
19         context.write(nameKey, urlValue);
20     }
21
22 }
23
```

UrlMapper.java | SubscriberMapper.java | UrlReducer.java | SubscriberReducer.java | CountUsersMapper.java

```
1
2  import java.io.IOException;
7
8
9
10 public class UrlReducer extends Reducer<Text, Text, Text, IntWritable>{
11
12     @Override
13     public void reduce(Text key, Iterable<Text> values, Context context) throws InterruptedException, IOException
14     {
15         int numberOfUrls =0 ;
16         for(Text v : values)
17         {
18             numberOfUrls++;
19         }
20
21         context.write(key, new IntWritable(numberOfUrls));
22     }
23
24 }
25
```

UrlMapper.java | SubscriberMapper.java | UrlReducer.java | SubscriberReducer.java | CountUsersMapper.java

```java
import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;


public class CountUsersMapper extends Mapper<LongWritable, Text, Text, Text> {

    public void map(Text key, Text value, Context context)throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] nameAndUrl = line.split("\t");

        Text urlKey = new Text(nameAndUrl[1]);
        Text userValue = new Text(nameAndUrl[0]);

        context.write(urlKey, userValue);
    }

}
```

Performance Tuning

TOOLS

- To check how hive translates hivequeries to MR jobs EXPLAIN command can be used. The hive optimization can be done by checking where all map reduce jobs stages can be optimized.
- With EXPLAIN the output will be
- ABSTRACT SYNTAX TREE: (TOK_QUERY
- (TOK_FROM (TOK_TABREF (TOK_TABNAME onecol))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
- (TOK_SELEXPR
  (TOK_FUNCTION sum (TOK_TABLE_OR_COL number))))))

- The stages(merge, MapReduce job)can be inferred from this output.

LIMIT TUNING

- LIMIT clause: executes entire query then only returns the cropped results.
- To avoid executing entire query following property can be used
- <property>
  <name>hive.**limit**.optimize.enable</name> <value>**true**</value>
  <description>Whether **to** enable **to** optimization **to** try a smaller subset **of data for simple LIMIT first**.</description>

  </property>
- NOTE: This will yield correct results only when the query has map steps. If the query also has reduce step the results can be incorrect. (Because the hive will not process all the data and will limit the records right at the starting of execution of jobs)

Join Optimization

- When joining 3 or more tables, if every ON clause uses the same join key, single MR job will be used.
- Hive assumes that the last table in the query is the largest. It will buffer other tables and stream the last table, while performing the joins. Therefore the largest table should be kept last in the query

PARALLEL EXECUTION

- Since hive queries are executed in stages, by default it executes one stage at a time.
- Some times jobs consist of stages that are not dependent on each other and can be executed in parallel, allowing overall job to complete quickly. For this following property can be set

<property>
<name>hive.exec.parallel**</name>**
<value>true**</value>**
<description>Whether to execute jobs in parallel**</description>**

</property>

JVM REUSE

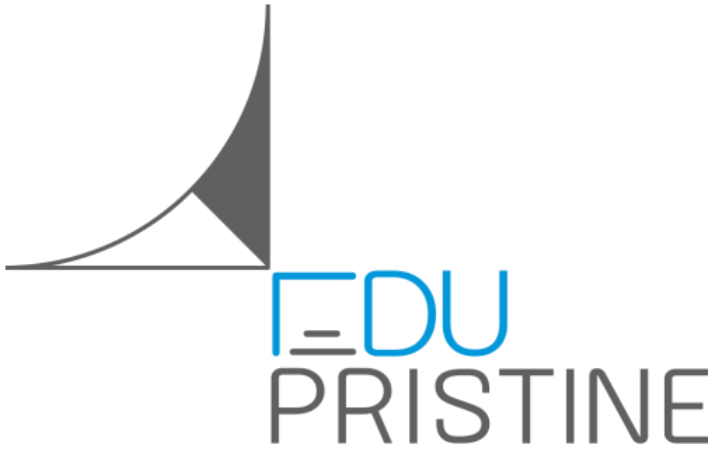
- When there are small number of files and lots of tasks most with short execution times.
- Default configuration of hadoop will typically launch map/reduce tasks in forked JVM. JVM startup has it's own overhead of starting. This can become bottleneck if there are thousands of tasks. To allow reuse of existing JVMs to a limit following property can be used

<property>
<name>mapred.job.reuse.jvm.num.tasks**</name>**
<value>10**</value>**
<description>How many tasks to run per jvm. If set to -1, there is no limit. **</description>**
</property>

Limitations of HIVE

Limitations of Hive

- Hive can’t be used for the problems that require response in milliseconds. Since hive launces MR job for most of the queries this staging takes time
- HIVE Queries are structural, you have to clearly specify what needs to be performed in what order using subqueries. This can become tedious if there are complex number of steps to be executed. For this Pig can be used which is more procedural like language, where the sequence of steps to be performed can be mentioned with individual statements
- Once data is inserted into table, data can’t be updated
- Splitting data: Data at various stages is merged and combined by SQL and in the end a single result is obtained. Hence splitting is not possible. Pig on the other hand can be used to save the result at different stages.



Thank You!

help@edupristine.com

www.edupristine.com

© **EduPristine** – www.edupristine.com