HIVE

What is HIVE?

- Data warehouse system for hadoop
- Runs SQL like queries that get compiled and run as Map Reduce jobs.
- Displays the result back to the user
- Data in Hadoop even though stored in unstructured has some vague structure associated with it

Why HIVE?

- To make the use of Hadoop file system and Map Reduce for non-developers. Users like scientist, analysts etc., don't need to know SQL syntax
- Writing SQL is faster than writing code

HIVE features

- Create table, create view, create index – DDL
- Select, where clause, group by, order by and joins – DML
- Pluggable input output format
- Pluggable
  - User Defined Functions – UDFs
  - User Defined Aggregate Functions – UDAF
  - User Defined Table Functions – UDTF
- Pluggable Serializable Deserializable – SerDes

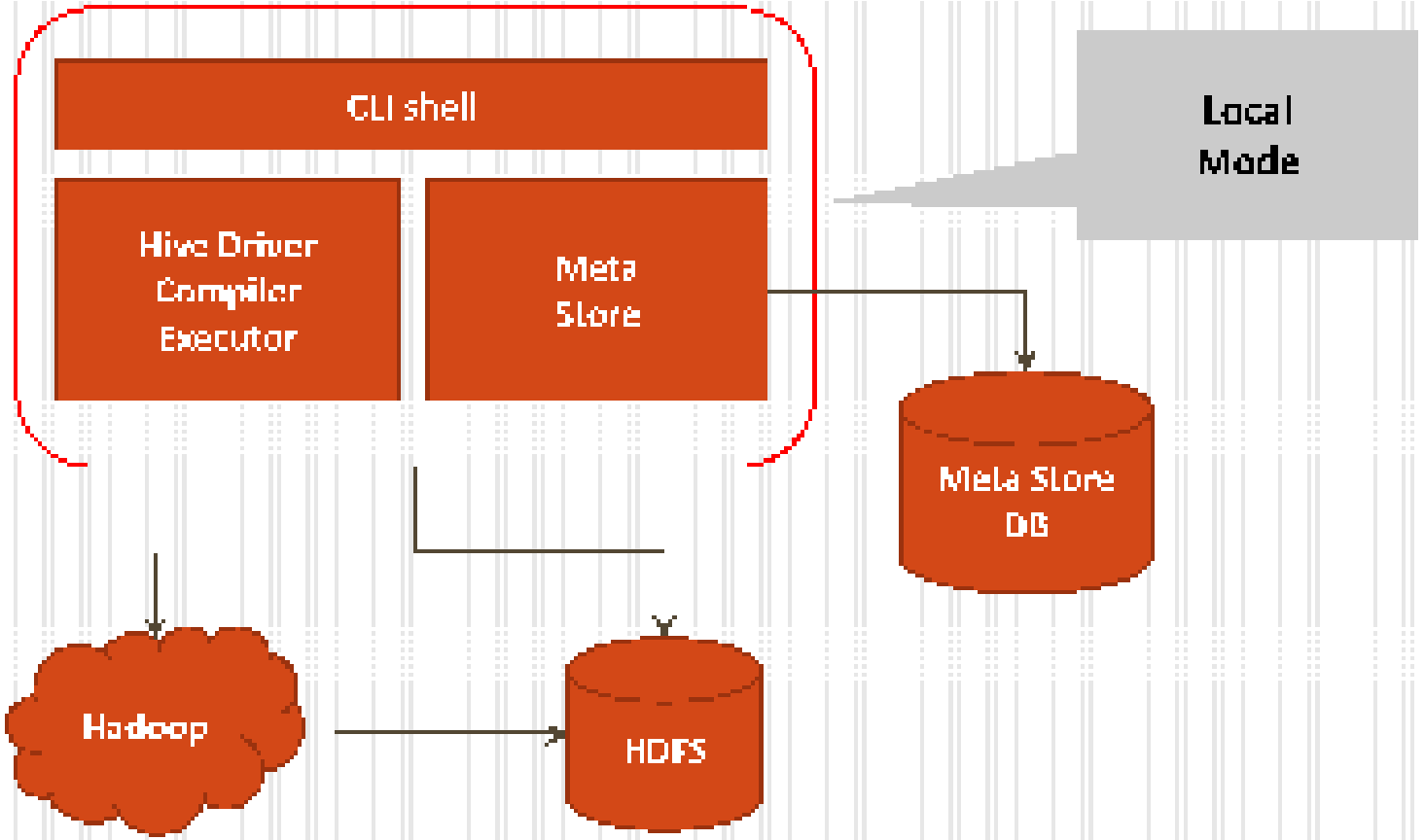
What HIVE is not ?

- It is not RDBMS
- OLTP workloads – low latency
- Correlated sub queries
- 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 MetaStore

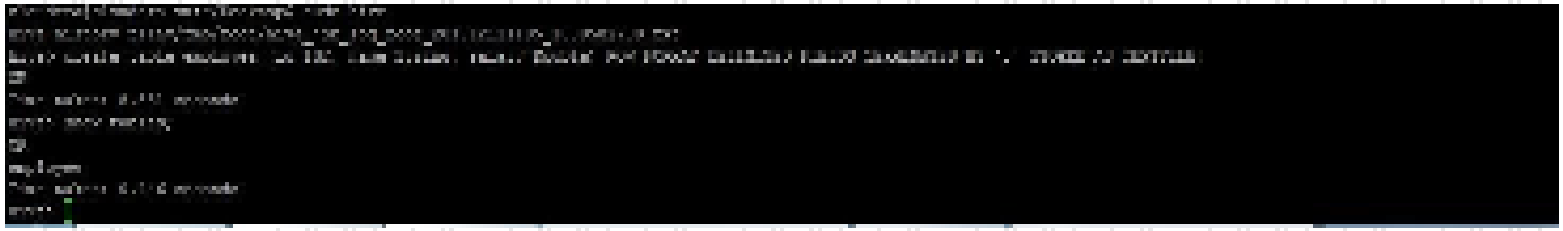
- Persists Schema, i.e. table definitions (names, columns, types)
- Location of table files
- Row format of table files
- Storage format of table files

How to create table in Hive ?

- CREATE TABLE [column name DATATYPE,...]
  - ROW FORMAT DELIMITED
  - FIELDS TERMINATED BY char
  - STORED AS TEXTFILE|SEQUENCEFILE|...

Example create table

- Hive> create table employee (id INT, name String, salary Double) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;



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.

Troubleshoot hive

```
hive> show databases;
FAILED: Error in metadata: javax.jdo.JDOFatalDataStoreException: Failed to start
 database '/home/cloudera/Desktop/metastore/metastore_db', see the next exceptio
n for details.
NestedThrowables:
java.sql.SQLException: Failed to start database '/home/cloudera/Desktop/metastor
e/metastore_db', see the next exception for details.
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTa
sk
hive> quit;
root@cloudera-vm:/home/cloudera# ls
cloudera  derby.log  imp.jar  newfile       prism         prism.jar  src.jar
data      Desktop    my.jar   part-r-00000  prismtmp.jar  sqoop.jar
root@cloudera-vm:/home/cloudera# cd Desktop/
root@cloudera-vm:/home/cloudera/Desktop# ls
derby.log  Learn Hadoop.desktop  metastore  prismtmp  README.txt
root@cloudera-vm:/home/cloudera/Desktop# cd metastore/
root@cloudera-vm:/home/cloudera/Desktop/metastore# ls
metastore_db
root@cloudera-vm:/home/cloudera/Desktop/metastore# cd metastore_db/
root@cloudera-vm:/home/cloudera/Desktop/metastore/metastore_db# ls
dbex.lck  db.lck  log  seg0  service.properties  tmp
root@cloudera-vm:/home/cloudera/Desktop/metastore/metastore_db# rm -rf *.lck
root@cloudera-vm:/home/cloudera/Desktop/metastore/metastore_db# sudo hive
Hive history file=/tmp/root/hive_job_log_root_201312111223_1902602719.txt
hive> show databases;
OK
default
Time taken: 7.267 seconds
hive> show tables;
OK
employee
Time taken: 0.561 seconds
```

Check table

```
hive> show tables;
OK
employee
Time taken: 0.561 seconds
hive> describe employee;
OK
id          int
name        string
salary      double
Time taken: 0.692 seconds
hive>
```

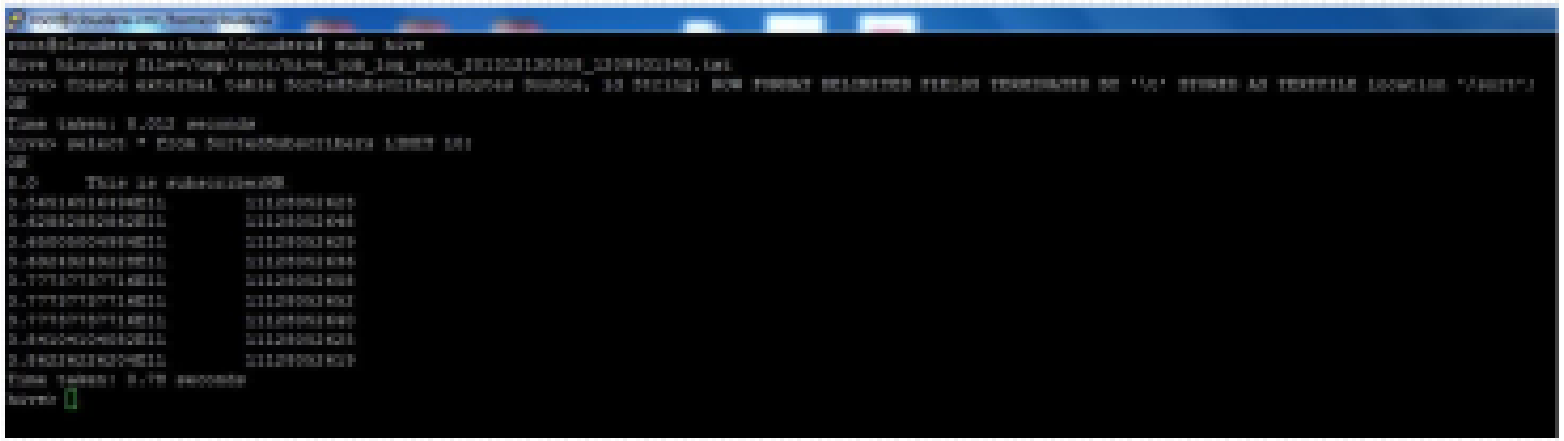
External table

- If the user wants to change the default directory to some specific existing directory in HDFS, the following command can be used
- Create external table if not exists [db_name.]table_name [(col_name data_type [COMMENT col_comment], ...)] [COMMENT table_comment] [ROW FORMAT row_format] [STORED AS file_format] [LOCATION hdfs_path]

Case study

- Create external table sortedata and base on sorted output (done in MR exercise).
- Display all the data using select * statement

Case study 1



Data types

- Spark column data types also have all the standard SQL data types as well as complex data types like Maps, Arrays, Struct

Syntax of collections

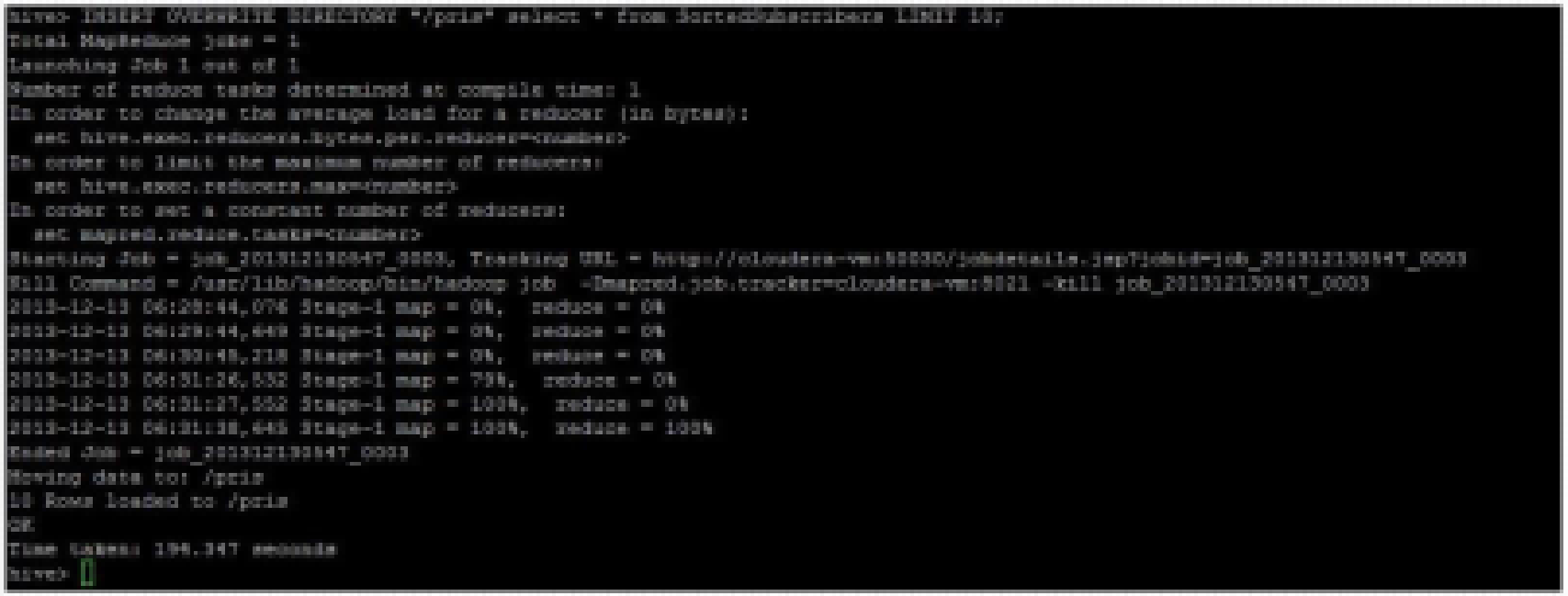
- Create table employee(id int, name string, salary Double, skills ArrayofString)
  - ROW FORMAT DELIMITED
  - FIELDS TERMINATED BY ','
  - COLLECTION ITEMS TERMINATED BY '|'
  - STORED AS TEXTFILE

Delete table

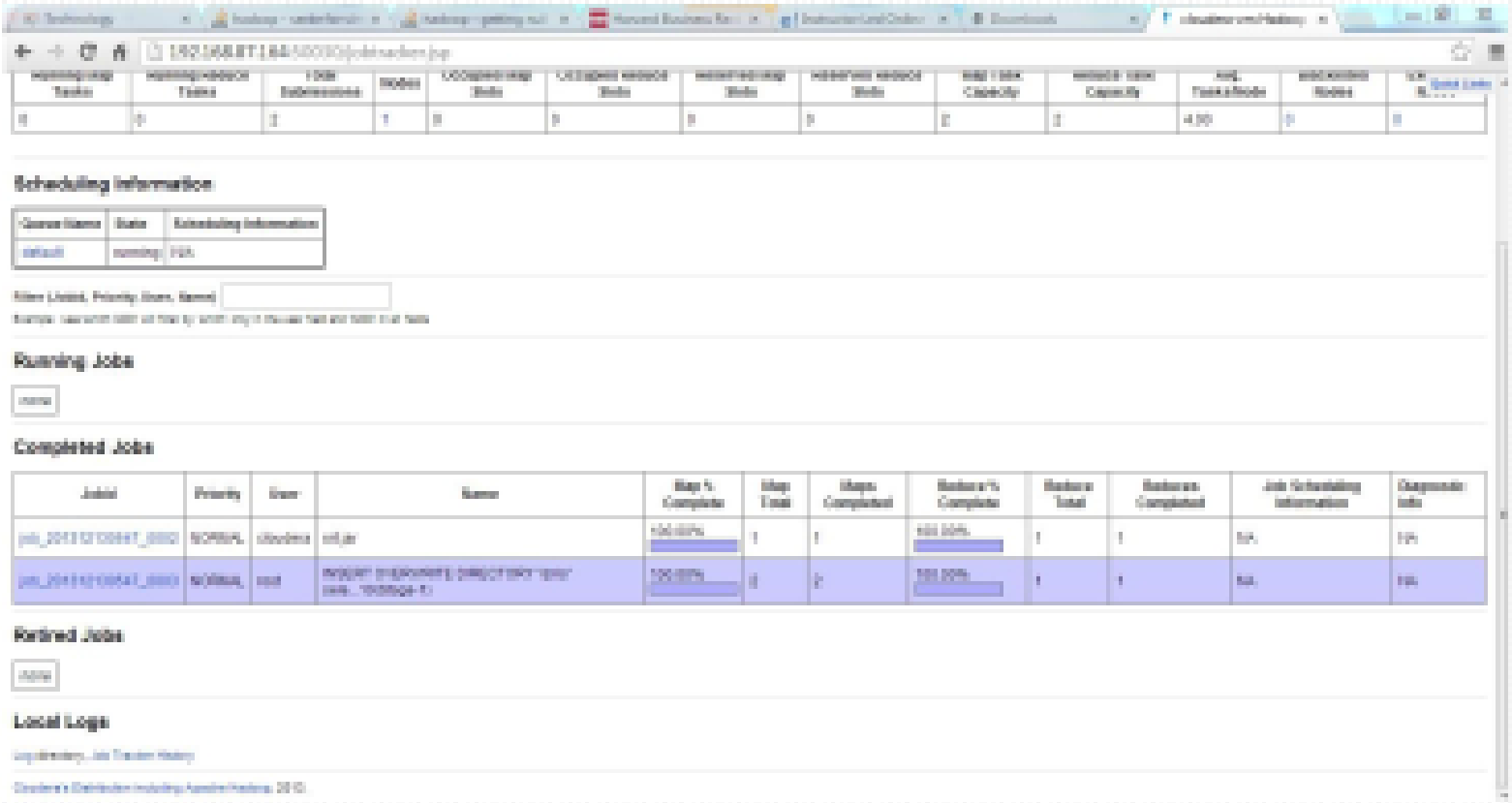
- Dropping table deletes table's directory
- Dropping external table only deletes the metadata not the directory

Writing output of query

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



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



Multiple DBs in Hive

- Default database is named default

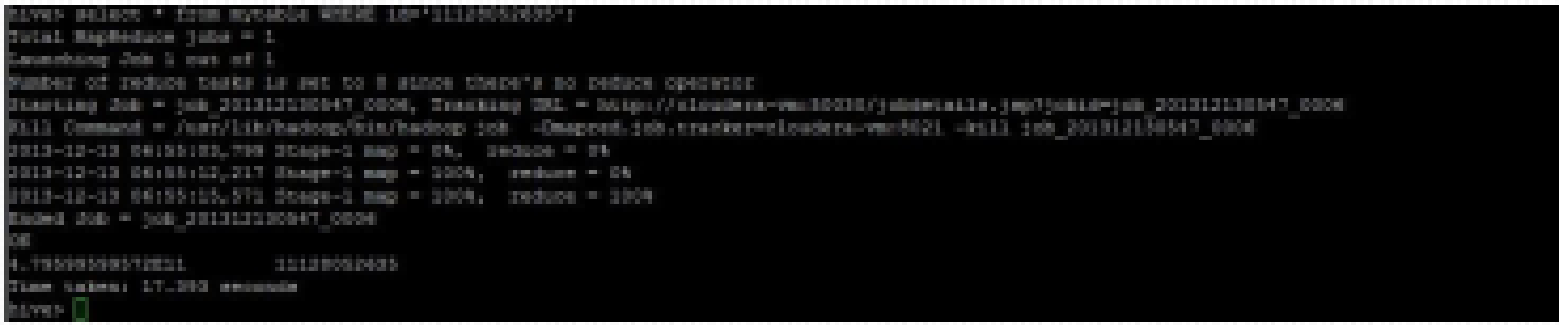
```
root@cloudera-vm:/home/cloudera# sudo hive
Hive history file=/tmp/root/hive_job_log_root_201312130641_1913664056.txt
hive> show databases;
OK
default
Time taken: 21.884 seconds
hive> create database primtime;
OK
Time taken: 2.147 seconds
hive> use primtime;
OK
Time taken: 0.074 seconds
hive> show 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: 0.264 seconds
hive>
```

HIVEQL

- Interface similar to SQL 92. However, some constructs are different
- Supports join, sub queries etc.
- Does not support UPDATE OR DELETE

Select statement

- Select expr1, expr2 .. FROM tablename t
- Where clause also supported



Order 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.

Sort by

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

```
Select expr, expr, ... from tablename a
           where condition
           sort by expr
```


USA : +1-908-366-7933
Skype ID : keylabstraining
Hyderabad : +91-9550645679
Bangalore : +91-8884584163
info@keylabstraining.com

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 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	...s
2	Fred	...
3	Jack	N
4	Tony	...

Employee Department

Emp ID	Department
1	IT
2	H
3	Fin
4	Admin

Inner join

- Select * from employee join employee department on employee.empid=employee department.empid;

EMP ID	Emp Name	Address	Emp ID	Department
1	[illegible]	[illegible]	1	[illegible]
2	[illegible]	[illegible]	2	[illegible]
3	[illegible]	[illegible]	3	[illegible]
4	[illegible]	[illegible]	4	[illegible]

Left outer join

- Select e.empid, e.empName, e.department from employee e left outer join employee e1 on (e.empid=e1.empid);

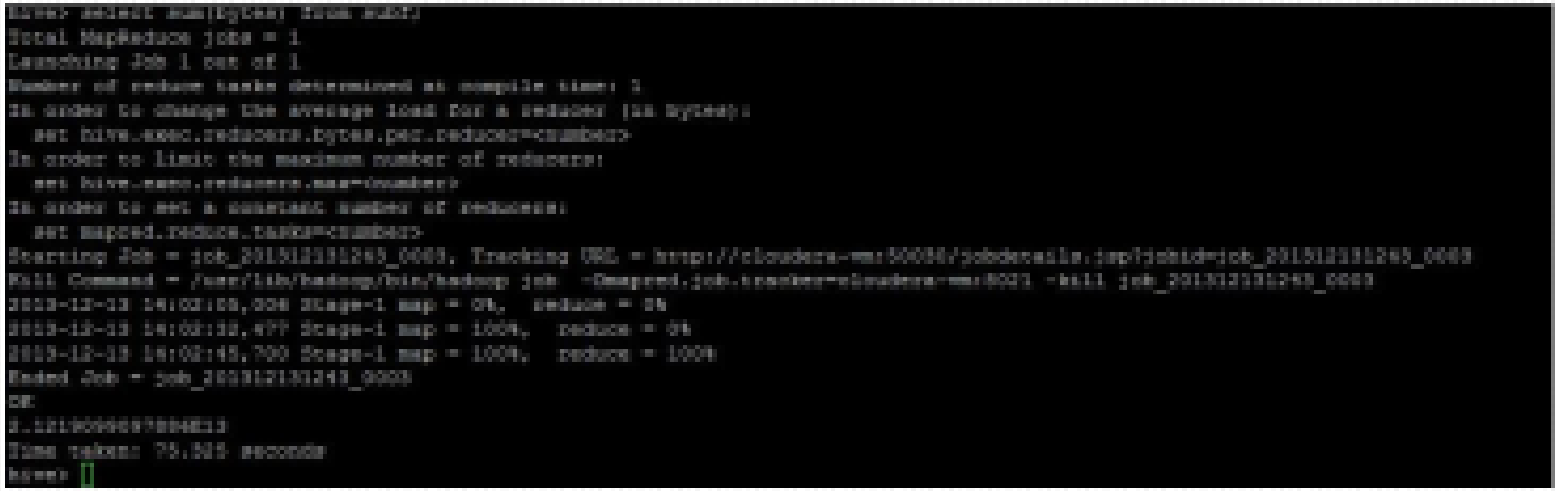
EMP ID	Emp Name	Department
1	[illegible]	IT
2	[illegible]	[illegible]
3	[illegible]	[illegible]
4	[illegible]	[illegible]
5	[illegible]	[illegible]

Hive functions

- round()
- floor()
- ceil()
- rand()
- exp()
- ln()
- log10()
- sqrt()
- abs()
- sin()

etc

Use of functions



Views

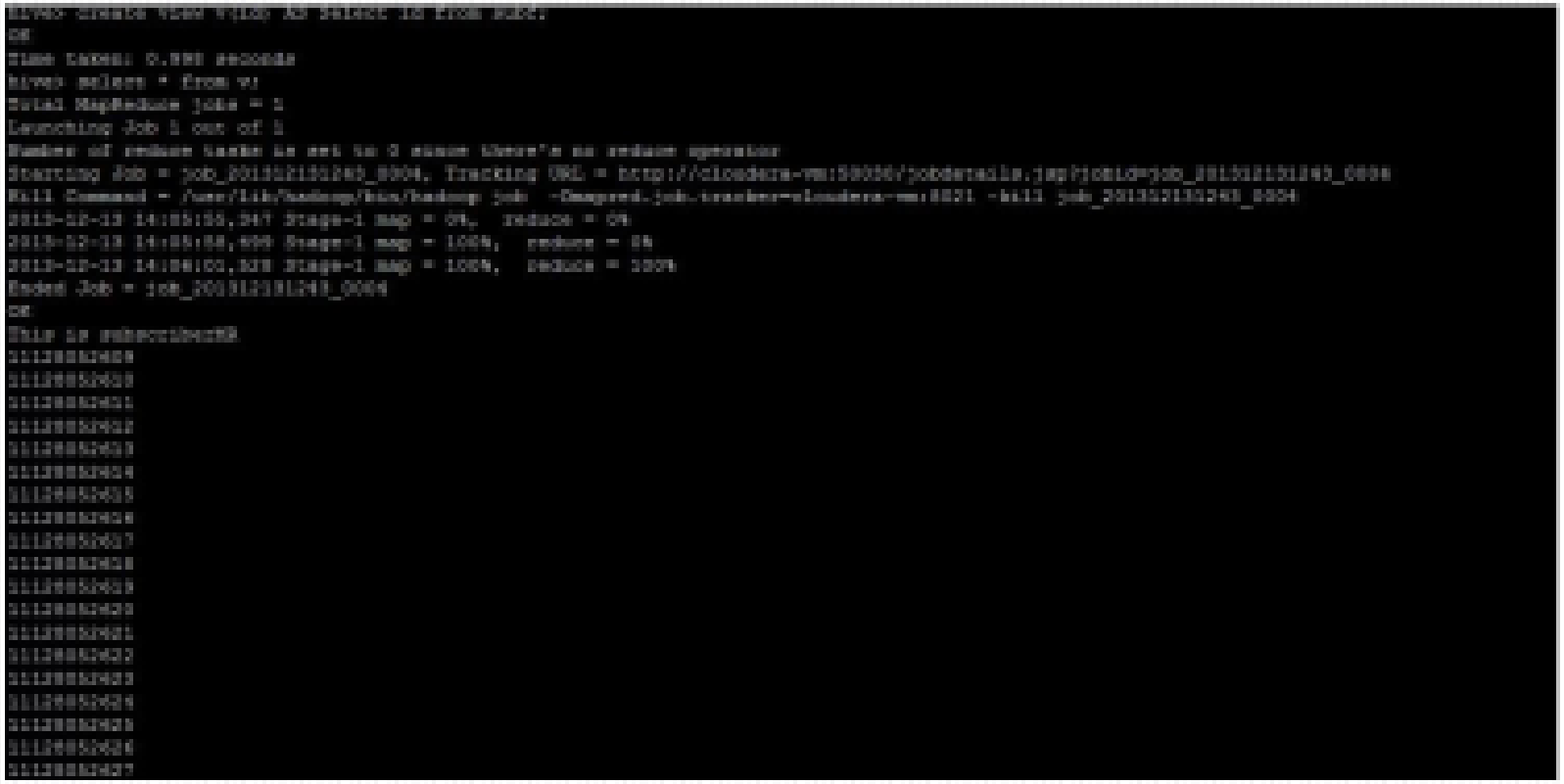
- CREATE VIEW view[column Names]

  As Select ...

To remove view

Drop View view

View



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

Use of partitioning

- Allows Hive to filter data at input path
- For example logs of currentdate.com (each study team's URL) can be stored for each day in a directory based on date.

Create partitioning table

- Create table t(column_spec)
  Partitioned by (column_name datatype)
  ROW_FORMAT

  Insert data into partition table
  Load data inpath [illegible]
  Into table t
  Partition(col=val);

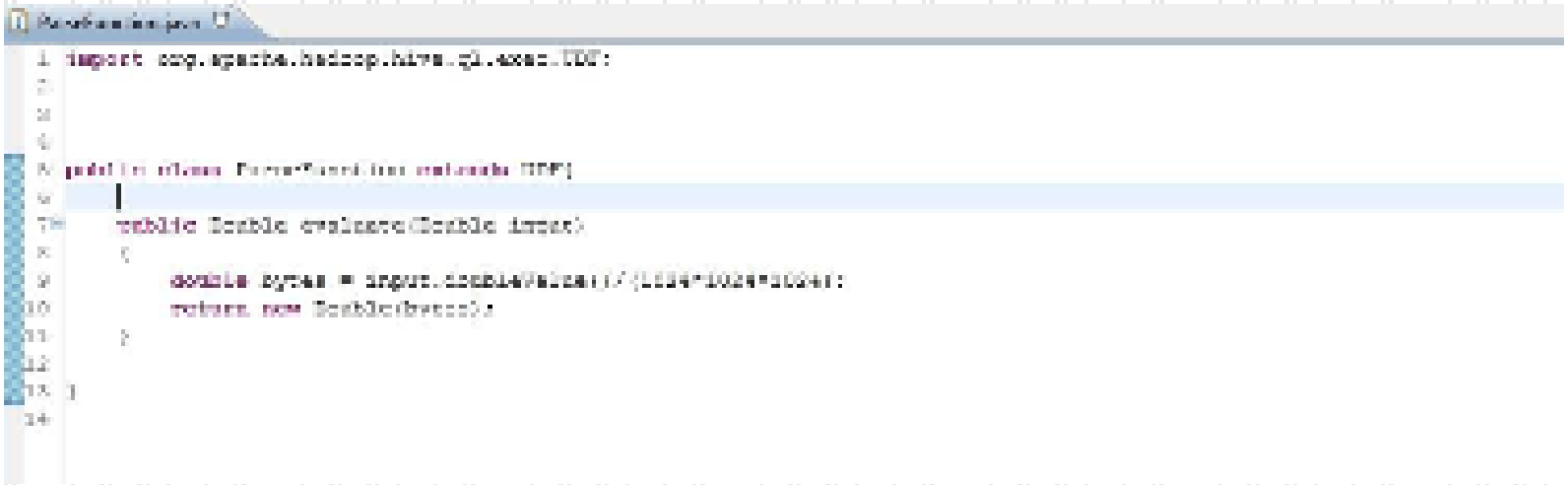

```
Hive history file=/tmp/root/hive_job_log_root_201312131440_820883879.txt
hive> load data inpath '/user' overwrite into table subscriberpart partition(tabledate='2014-01-06');
Loading data to table default.subscriberpart partition (tabledate=2014-01-06)
OK
Time taken: 8.825 seconds
hive> quit;
```

```
root@cloudera-vm:/home/cloudera# hadoop dfs -ls /user/hive/warehouse/
Found 3 items
drwxr-xr-x   - root supergroup          0 2013-12-13 06:42 /user/hive/warehouse/pristine.db
drwxr-xr-x   - root supergroup          0 2013-12-13 14:31 /user/hive/warehouse/subscriberlogs
drwxr-xr-x   - root supergroup          0 2013-12-13 14:41 /user/hive/warehouse/subscriberpart
root@cloudera-vm:/home/cloudera# hadoop dfs -ls /user/hive/warehouse/subscriberpart/
Found 1 items
drwxr-xr-x   - cloudera supergroup          0 2013-12-13 14:40 /user/hive/warehouse/subscriberpart/tabledate=2014-01-06
```

Creating custom UDFs

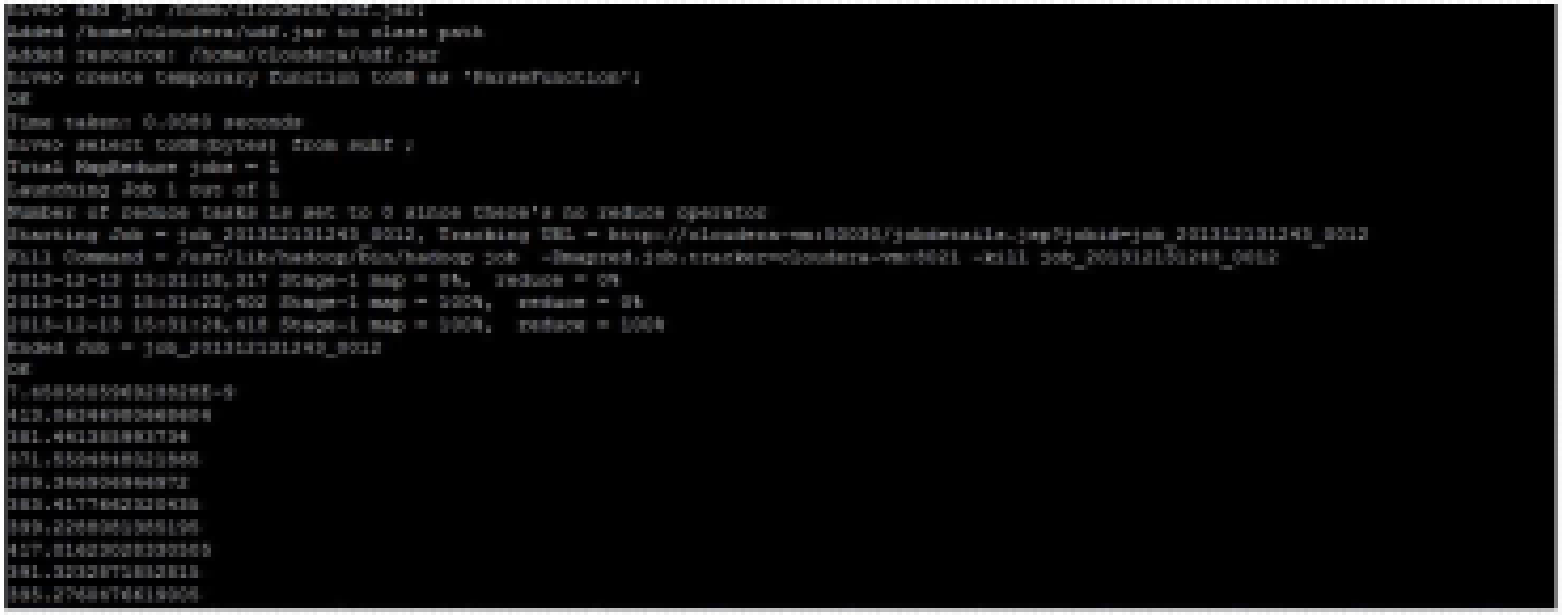
- To implement custom UDF, need to extend org.apache.hadoop.hive.ql.exec.UDF class present in hive-exec-0.7.0-cdh3u0. This jar can be copied from /usr/lib/hive/lib of VM.
- The evaluate method can be implemented accordingly

Custom UDF



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



public class PowerFunction extends UDF{

    public Double evaluate(Double input)
    {
        double power = input.doubleValue()*input.doubleValue();
        return new Double(power);
    }

}
```

How to run UDF



Serde

- Serde (Serializer/Deserializer) is used by Hive to control how rows are read and written to/from files
- When used as serializer (i.e. insert) tables 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. Uses a 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.binarysortable	Optimized for storing data in order at the cost of compactness
ColumnarSerDe	org.apache.hadoop.hive.serde2.columnar	For storing data in RCFile format
RegexSerDe	org.apache.hadoop.hive.contrib.serde2	For textual data where columns are specified by regular expression

Storage format

- 2 dimensions that control table 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

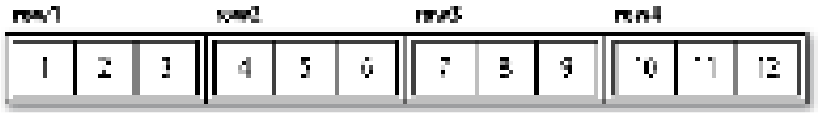
- Default format is delimited text with a row per line
- Rows in a table are delimited by newline character
- Hive uses LazySimpleSerDe for this format

Binary storage format RCFile and sequencefile

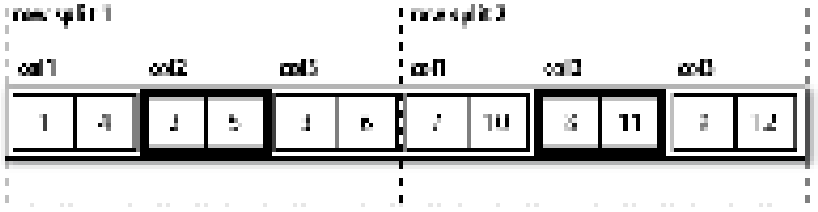
- Using case study 1 of MR the output is generated by Subscriber Mapper and Reducer was sequence file (key/value pairs)
- Sequence files are row oriented
- RCFiles, row columnar file. Similar to sequence file but data is stored in column oriented fashion

Logical table

Row-oriented layout (Sequence File)



Column-oriented layout (RCFile)



Performance Tuning

TOOLS

- To check how hive translates the queries 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(mapReduce jobs) 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 in a query ON clause uses the same join key, single MR job will be used.
- Hive assumes the last table in the query is the largest. It will buffer other tables and stream the last table so it is performing the joins. Therefore the largest table should be kept last in the query.

PARALLEL EXECUTION

- Hive query executes one stage at a time by default. It executes one stage at a time.
- Some Hive jobs consist of stages that are not dependent on each other and can be executed in parallel, allowing those jobs to complete quickly. For this the 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 (maps) with short execution time.
- Default configuration of Hadoop will typically launch map/reduce tasks in forked JVM. JVMs start up has it's own overhead of starting. This can become bottleneck if there are thousands of tasks. To allow reuse of existing JVMs to 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 launches MR jobs for most of the queries, this staging takes time.
- HIVE Queries are structured – 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 a 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.