The Hive

- Presented By

Siva Kumar Bhuchipalli

CONFIDENTIAL

Contents:

Pig VS Hive

Employees.txt file is there id, name, age, deptid

```
LOAD 'Employees.txt' USING PigStorage(',') AS ();
A = FOREACH emp GENERATE name, deptid, age;
A1 = FILTER A BY age > 30;
B = GROUP A1 BY deptid;
C = FOREACH B GENERATE group, A1.name;
DUMP C;
```

```
SELECT name, deptid FROM emp WHERE age > 30 group by deptid;
```

1. No of lines get reduced
2. No Need of carrying so many aliases
3. Pig script scope is just for that session whereas hive table is persistent across the sessions
4. Most of the industry programmers might already be familiar SQL syntax whereas pig might be a new tool to learn
5. Every problem you can solve in Pig can be solved in Hive
6. Hive Performance is also around the same range with Pig
7. You can achieve extra functionalities with Hive Metastore, JDBC Clients to connect to reporting tools

Hive

Hive is an Important Tool in the hadoop ecosystem and it is framework for data warehousing on top of hadoop.

Hive is initially Developed at Facebook but now its is an Open-source Apache project.

What HIVE Provides?

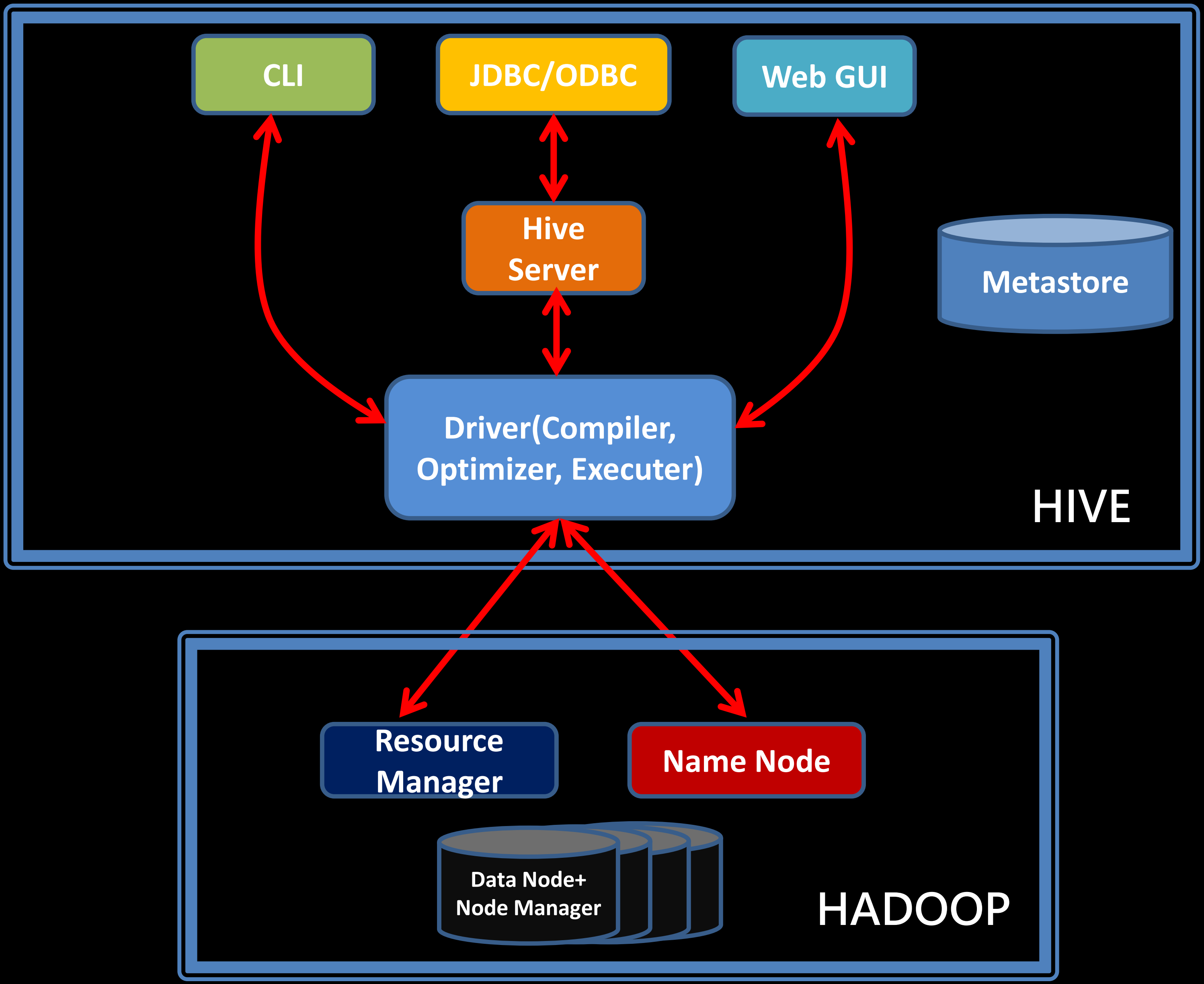
- Tools to enable easy data ETL (Extract /transform/Load).
- A mechanism to project structure on a variety of data formats.
- Access to file stored either directly in HDFS or other data storage system as HBASE.
- Query execution through MapReduce jobs.
- SQL like language called HiveQL that facilitates querying and managing large data sets residing on Hadoop.

What HIVE Provides?

RDBMS VS HIVE

❖ RDBMS is a Database.	❖ HIVE is a Data warehouse.
❖ RDBMS supports schema on write time.	❖ HIVE supports schema on read time.
❖ Read and Write Many times.	❖ Write once and Read Many times.
❖ Record level Insertion, Updates and deletes is possible.	❖ Record level Insertion, Updates and deletes is not possible.
❖ Maximum data size allowed will be 10s of Terabytes.	❖ Maximum data size allowed will be 100s of Petabytes.
❖ RDBMS is suited for the dynamic data analysis.	❖ HIVE is suited for the static data analysis
❖ OLTP	❖ OLAP

Hive Architecture:



Major Components of Hive :

❖ **UI :**

UI means User Interface, The user interface for users to submit queries and other operations to the system.

❖ **Driver :**

The Driver is used for receives the quires from UI .This component implements the notion of session handles and provides execute and fetch APIs modelled on JDBC/ODBC interfaces.

❖ **Compiler :**

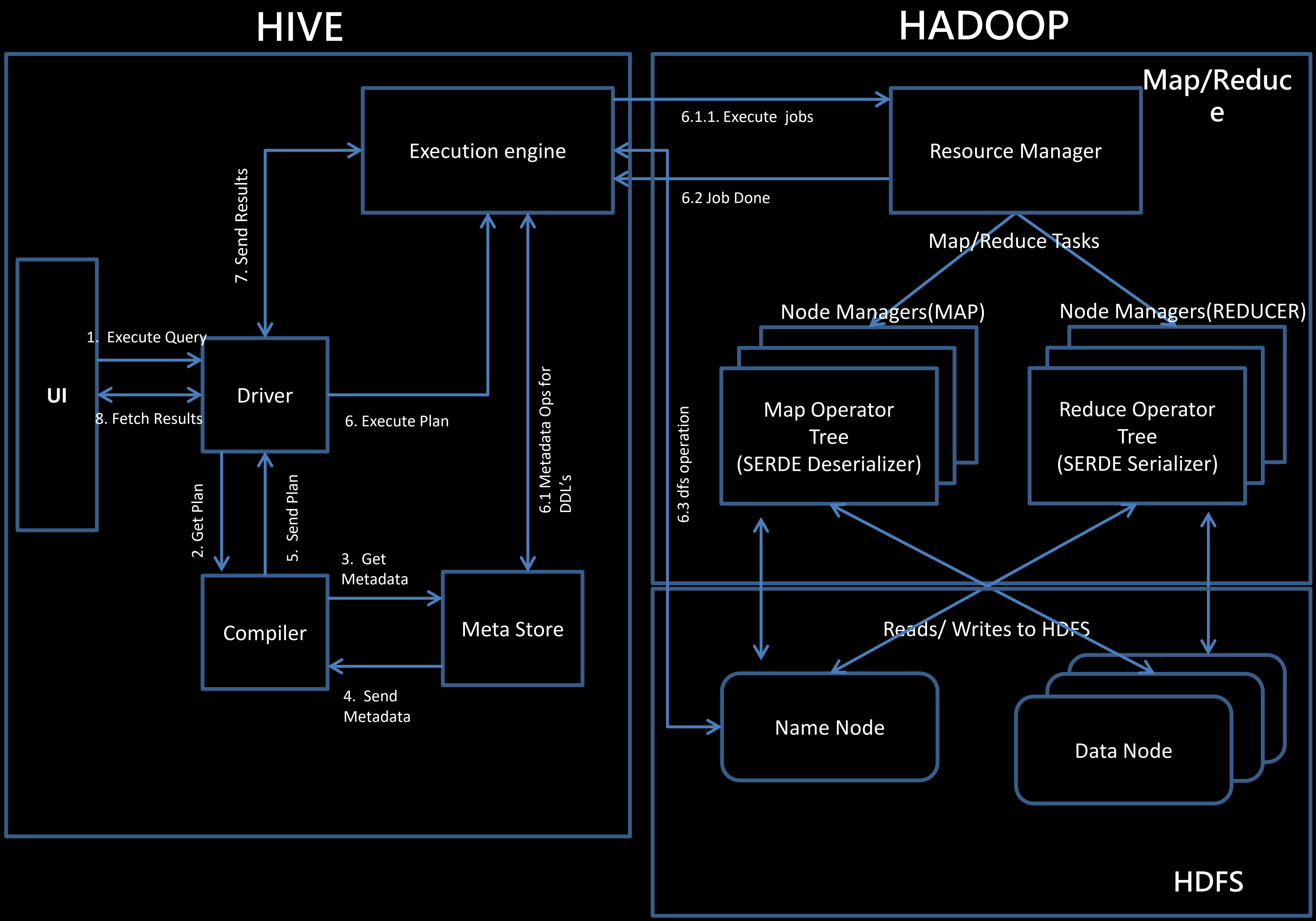
The component that parses the query, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.

❖ **MetaStore :**

The component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding HDFS files where the data is stored.

❖ **Execution Engine :**

The component which executes the execution plan created by the compiler. The plan is a DAG of stages. The execution engine manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components.


HIVE
HADOOP
Map/Reduce
Execution engine
Resource Manager
6.1.1. Execute jobs
6.2 Job Done
7. Send Results
Map/Reduce Tasks
Node Managers(MAP)
Node Managers(REDUCER)
1. Execute Query
UI
Driver
8. Fetch Results
6. Execute Plan
6.1 Metadata Ops for DDL's
6.3 dfs operation
Map Operator Tree (SERDE Deserializer)
Reduce Operator Tree (SERDE Serializer)
2. Get Plan
5. Send Plan
3. Get Metadata
Compiler
Meta Store
4. Send Metadata
Reads/ Writes to HDFS
Name Node
Data Node
HDFS

Step 1 :

The UI calls the execute interface to the Driver.

Step 2 :

The Driver creates a session handle for the query and sends the query to the compiler to generate an execution plan.

Step 3&4 :

The compiler needs the metadata so send a request for get Meta Data and receives the send Meta Data request from Meta Store.

Step 5 :

This metadata is used to type check the expressions in the query tree as well as to prune partitions based on query predicates. The plan generated by the compiler is a DAG of stages with each stage being either a map/reduce job, a metadata operation or an operation on HDFS. For map/reduce stages, the plan contains map operator trees (operator trees that are executed on the mappers) and a reduce operator tree (for operations that need reducers).

Step 6 :

The execution engine submits these stages to appropriate components (steps 6, 6.1, 6.2 and 6.3). In each task (mapper/reducer) the deserializer associated with the table or intermediate outputs is used to read the rows from HDFS files and these are passed through the associated operator tree. Once the output generate it is written to a temporary HDFS file through the serializer. The temporary files are used to provide the to subsequent map/reduce stages of the plan. For DML operations the final temporary file is moved to the table's location

Step 7&8 :

For queries, the contents of the temporary file are read by the execution engine directly from HDFS as part of the fetch call from the Driver

Hive + SQL

Relational Database uses SQL as their Query Language.

If data warehouses are moved to hadoop then users of these data warehouses must learn new language and tools to become productive on hadoop data.

“Instead of this Hive Provide HQL which is similar to SQL”

HQL

Hive Database Commands

Create Database

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
 [COMMENT database_comment]
 [LOCATION hdfs_path]
 [WITH DBPROPERTIES (property_name=property_value, ...)];
```

- IF NOT EXISTS – Optional, if a database with same name already exists, then it will not try to create it again and will not show any error message.
- COMMENT – It is also optional. It can be used for providing short description
- LOCATION – It is also optional. By default all the hive databases will be created under default warehouse directory as /user/hive/warehouse/*database_name.db* .
- But if we want to specify our own location then this option can be specified.
- DBPROPERTIES – Optional but used to specify any properties of database in the form of (key, value) separated pairs.

Hive Database Commands

Create Database Examples

```
CREATE DATABASE IF NOT EXISTS test_db
COMMENT "Test Database created for tutorial"
WITH DBPROPERTIES(
'Date' = '2014-12-03',
'Creator' = ‘Siva B',
'Email' = ‘siva@somewhere.com'
);
```

Show Databases

```
SHOW (DATABASES|SCHEMAS) [LIKE identifier_with_wildcards];

hive> show databases;
hive> SHOW DATABASES LIKE '*db*';
```

Use Databases

```
hive> USE database_name;
Hive> set hive.cli.print.current.db=true;
```

Hive Database Commands

Describe Databases

```
hive> (DESCRIBE|DESC) (DATABASE|SCHEMA) [EXTENDED] database_name;

hive> DESCRIBE DATABASE test_db;
hive> DESCRIBE DATABASE EXTENDED test_db;
```

Alter Databases

```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES
(property_name=property_value, ...);

ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;

hive> ALTER SCHEMA test_db SET DBPROPERTIES ('Modified by' = 'siva');
```

Drop Databases

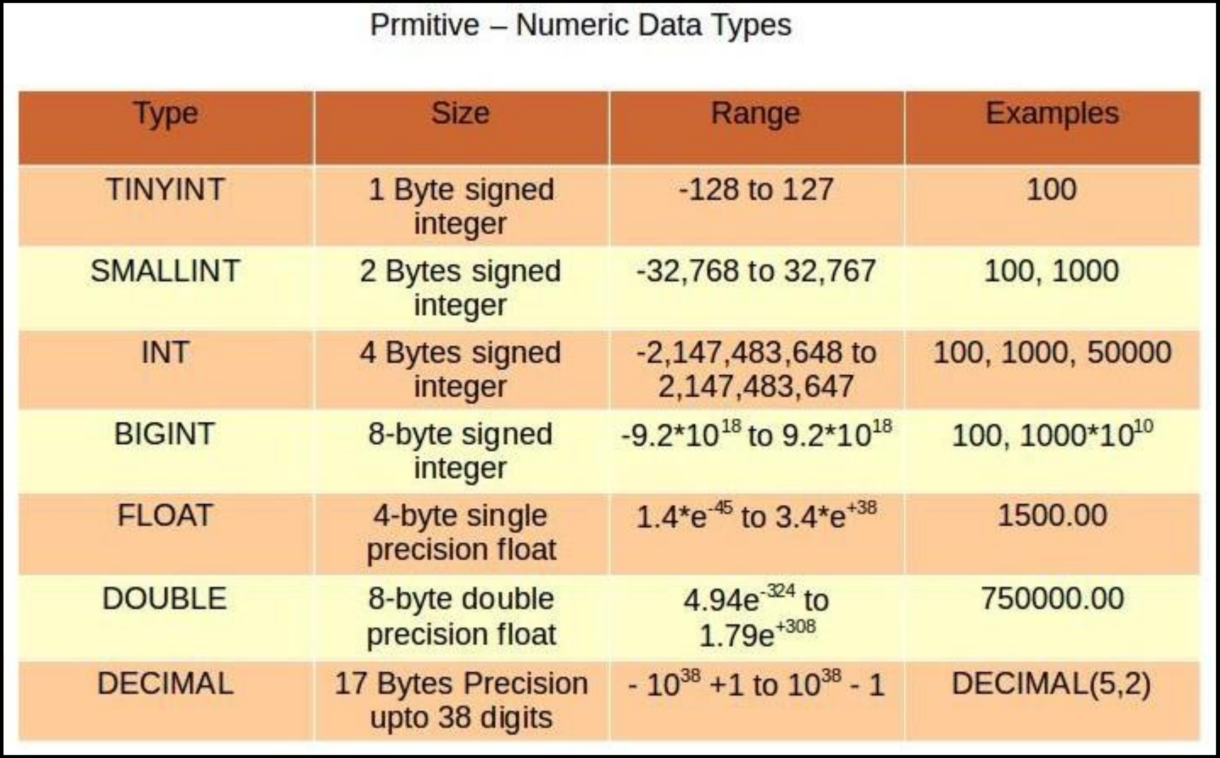
```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE]
```

RESTRICT – Optional and even if it is used, it is same as default hive behavior, i.e. it will not allow database to be dropped until all the tables inside it are dropped.
CASCADE – Allows to drop the non-empty databases. DROP with CASCADE is equivalent to dropping all the tables separately and dropping the database finally in cascading manner

Hive Data Types

Primary Data Types

- Numeric Types
- String Types
- Date/Time Types
- Miscellaneous Types

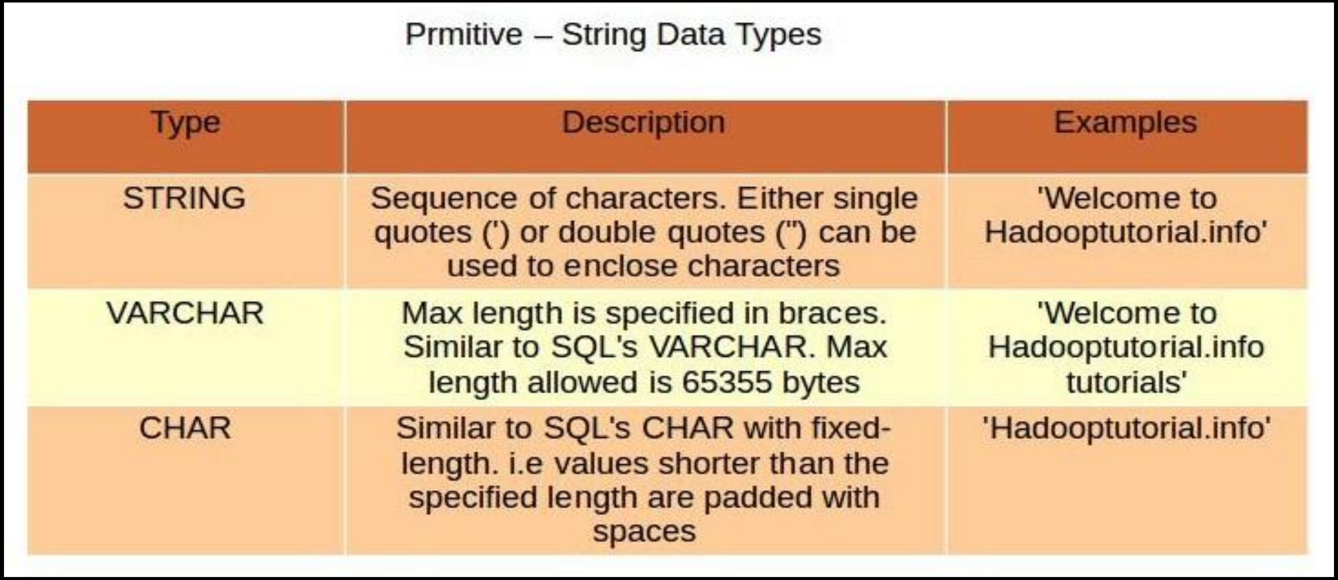


Prmitive – Numeric Data Types

Type	Size	Range	Examples
TINYINT	1 Byte signed integer	-128 to 127	100
SMALLINT	2 Bytes signed integer	-32,768 to 32,767	100, 1000
INT	4 Bytes signed integer	-2,147,483,648 to 2,147,483,647	100, 1000, 50000
BIGINT	8-byte signed integer	$-9.2*10^{18}$ to $9.2*10^{18}$	100, $1000*10^{10}$
FLOAT	4-byte single precision float	$1.4*e^{-45}$ to $3.4*e^{+38}$	1500.00
DOUBLE	8-byte double precision float	$4.94e^{-324}$ to $1.79e^{+308}$	750000.00
DECIMAL	17 Bytes Precision upto 38 digits	$-10^{38}+1$ to $10^{38}-1$	DECIMAL(5,2)

Hive Data Types

Primary Data Types



Prmitive – String Data Types

Type	Description	Examples
STRING	Sequence of characters. Either single quotes (') or double quotes (") can be used to enclose characters	'Welcome to Hadooptutorial.info'
VARCHAR	Max length is specified in braces. Similar to SQL's VARCHAR. Max length allowed is 65355 bytes	'Welcome to Hadooptutorial.info tutorials'
CHAR	Similar to SQL's CHAR with fixed-length. i.e values shorter than the specified length are padded with spaces	'Hadooptutorial.info'

DATE values are represented in the form YYYY-MM-DD.

Example: DATE ‘2014-12-07′.

Date ranges allowed are 0000-01-01 to 9999-12-31.

TIMESTAMP use the format yyyy-mm-dd hh:mm:ss[.f...].

Misc

BOOLEAN - stores true or false values

BINARY - An array of Bytes and similar to VARBINARY in many RDBMSs

Complex Data Types

Array

An Ordered sequence of similar type elements that are indexable using zero based integer. Similar to Array in Java.

Map

Element in the form of Key, Value collections separated by delimiter.
It is a Collection of Key-Value Pair

Struct

The collection of elements with Different Data types.

Delimiters in Table Data

Delimiter	Code	Description
\n	\n	Record or row delimiter
^A (Ctrl+A)	\001	Field delimiter
^B (Ctrl+B)	\002	Element delimiter in ARRAYs and STRUCTs
^C (Ctrl+C)	\003	Delimits key/value pairs in a MAP

Example Table Creation

```
 CREATE TABLE user (
    name      STRING,
    id        BIGINT,
    isFTE     BOOLEAN,
    role      VARCHAR(64),
    salary    DECIMAL(8,2),
    phones    ARRAY<INT>,
    deductions MAP<STRING, FLOAT>,
    address   STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>,
    others    UNIONTYPE<FLOAT,BOOLEAN,STRING>,
    misc      BINARY
    )
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\001‘
   COLLECTION ITEMS TERMINATED BY '\002'
   MAP KEYS TERMINATED BY '\003'
   LINES TERMINATED BY '\n';

```

Sample Data

```
Chandra,100,TRUE,Tech
Lead,25000.00,97888876:86555555,PF#1000.00,JubileeHills:Hyd:TG:500033,2:Chandra
Record,stringvalue
Teja,101,TRUE,Tech
Lead,25000.00,97888876:86555555,PF#1000.00,JubileeHills:Hyd:TG:500033,1:TRUE,stringvalue
Varshini,102,False,Dev,15000.00,97888876:86555555,PF#1000.00,JubileeHills:Hyd:TG:500033,0
:35.05,stringvalue
Neeraja,103,TRUE,Tech
Lead,25000.00,97888876:86555555,PF#1000.00,JubileeHills:Hyd:TG:500033,2:Neeraja
Record,stringvalue
```

Change Delimiters in Existing Table Data

```
ALTER TABLE ndx_metadata.dataset_char_value
SET SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe' WITH
SERDEPROPERTIES ('field.delim' = '\t');
```

Hive QUERY

Creating a table

```
hive> CREATE TABLE <table-name>
       (<column name> <data-type>,
       <column name> <data type>);
hive> CREATE TABLE <table-name>
       (<column name> <data-type>,
       <column name> <data type>)
       row format delimited fields terminated by '\t';
hive> CREATE TABLE events(a int, b string);
```

Loading data in a table

```
hive> LOAD DATA LOCAL INPATH '<input-path>'  INTO TABLE events;
hive> LOAD DATA LOCAL INPATH '<input-path>' OVERWRITE INTO TABLE events;
```

Viewing the list of tables

```
hive> show tables;
```

Different Load Types

– Load data from HDFS location

File is copied from the provided location to /user/hive/warehouse/ (or configured location)

```
hive> LOAD DATA INPATH '/training/hive/user-posts.txt'
> OVERWRITE INTO TABLE posts;
```

– Load data from a local file system

File is copied from the provided location to /user/hive/warehouse/ (or configured location)

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'
> OVERWRITE INTO TABLE posts;
```

– Utilize an existing location on HDFS

Just point to an existing location when creating a table

```
hive> CREATE TABLE posts
> (user STRING, post STRING, time BIGINT)  ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ',‘  STORED AS TEXTFILE
➢ LOCATION '/training/hive/';
```

```
➢ INSERT INTO TABLE posts SELECT * FROM another_table;
```

Hive QUERY

Displaying contents of the table

```
hive>  select * from <table-name>;
```

Dropping tables

```
hive> drop table <table-name>;
```

Altering tables

Table names can be changed and additional columns can be dropped:

```
hive> ALTER TABLE events ADD/REMOVE/CHANGE COLUMNS
   (new_col INT);
hive> ALTER TABLE events RENAME TO pokes;
```

Using WHERE Clause

The where condition is a boolean expression. Hive does not support IN, EXISTS or sub queries in the WHERE clause.

```
hive> SELECT * FROM <table-name> WHERE <condition>
```

Hive QUERY

Using Group by

```
hive> SELECT deptid, count(*) FROM department GROUP BY
   deptid HAVING deptid > 300;
```

Using Join

ATTENTION Hive users:

- Only equality joins, outer joins, and left semi joins are supported in Hive.
- Hive does not support join conditions that are not equality conditions as it is very difficult to express such conditions as a Map Reduce job.
- Also, more than two tables can be joined in Hive.

```
hive> SELECT a.* FROM a JOIN b ON (a.id = b.id)
Hive> SELECT a.val, b.val, c.val
       FROM a JOIN b ON (a.KEY = b.key1) JOIN c ON (c.KEY
   = b.key1)
```

MR Job Execution for Hive Queries

```
select * from user;                 // No MR Job
select deptid, name from dept;      // No MR Job
select deptid, name from dept where deptid > 100; // No MR Job
select count(*) from user;          // MR Job executed
select deptid, count(*) from user group by deptid;  // MR Job executed
select deptid, deptname, count(*) from user group by deptid,deptname; //MR
   Job
```

TRUNCATE TABLE table_name [PARTITION partition_spec];

Removes all rows from a table or partition(s). Currently target table should be managed table or exception will be thrown.

DESCRIBE FORMATTED TABLE

```
hive> describe formatted user;
OK
# col_name                      data_type                       comment

name                            string
id                              bigint
isfte                           boolean
role                            varchar(64)
salary                          decimal(8,2)
phones                          array<int>
deductions                      map<string,float>
address                         struct<street:string,city:string,state:string,zip:int>
others                          uniontype<float,boolean,string>
misc                            binary

# Detailed Table Information
Database:                       default
Owner:                          cloudera
CreateTime:                     Wed Dec 21 17:48:01 PST 2016
LastAccessTime:                 UNKNOWN
Protect Mode:                   None
Retention:                      0
Location:                       hdfs://quickstart.cloudera:8020/user/hive/warehouse/user
Table Type:                     MANAGED_TABLE
Table Parameters:
      COLUMN_STATS_ACCURATE     true
      numFiles                  1
      totalSize                 458
      transient_lastDdlTime     1482371532

# Storage Information
SerDe Library:                  org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:                    org.apache.hadoop.mapred.TextInputFormat
OutputFormat:                   org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:                     No
Num Buckets:                    -1
Bucket Columns:                 []
Sort Columns:                   []
Storage Desc Params:
      colelction.delim          :
      field.delim               ,
      line.delim                \n
      mapkey.delim              #
      serialization.format      ,
```

SHOW CREATE TABLE

```
Hive>show create table user;
OK
CREATE TABLE `user`(
  `name` string,
  `id` bigint,
  `isfte` boolean,
  `role` varchar(64),
  `salary` decimal(8,2),
  `phones` array<int>,
  `deductions` map<string,float>,
  `address` struct<street:string,city:string,state:string,zip:int>,
  `others` uniontype<float,boolean,string>,
  `misc` binary)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY ':'
  MAP KEYS TERMINATED BY '#'
  LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'hdfs://quickstart.cloudera:8020/user/hive/warehouse/user'
TBLPROPERTIES (
  'COLUMN_STATS_ACCURATE'='true',
  'numFiles'='1',
  'totalSize'='458',
  'transient_lastDdlTime'='1482371532')
```

Table Types

Managed Tables – Default table type in Hive

- Tables data is manged by Hive by moving data into its warehouse directory configured by **hive.metastore.warehouse.dir** (by default /user/hive/warehouse).
- If this table is dropped both **data** and **metadata** (schema) are **deleted**. I.e. these tables are owned by Hive.

External Tables

- These tables are not managed or owned by Hive.
- If these tables are dropped only the schema from metastore will be deleted but not the data files from external location.
- Provides convenience to share the tables data with other tools like Pig, HBase, etc…
- Simple query to change managed to External or vice-versa.

```
ALTER TABLE dataset_char_value SET TBLPROPERTIES('EXTERNAL'='FALSE')
```

Temporary Tables

- By the name itself, these are temporary and available till end of current session only.
- Useful in case of creating intermediate tables to copy data records from one table to another but can be deleted after our copy operation.

Metastore Types

Why we store metadata in RDBMS

- 1. To Support Alter command/modification of metadata;
- 2. To achieve faster access to metadata and as metadata is small in size and can be easily managed by RDBMS
- 3. RDBMS runs faster on small data

Embedded Metastore – Default Metastore type in Hive

- Derby database is the default RDBMS that ships with every Hive Installation
- javax.jdo.option.ConnectionURL →jdbc:derby:;databaseName=metastore_db;create=true
- Multi Users are not supported

Local Metastore

- Instead of Derby, metadata will be stored either in MySQL, Postgres or any other RDBMS
- This has support for multi user
- MySQL will be installed on the same machine from where hive session is being invoked

Remote Metastore

- This has support for multi user
- MySQL will be installed on the remote machine from where hive session is being invoked

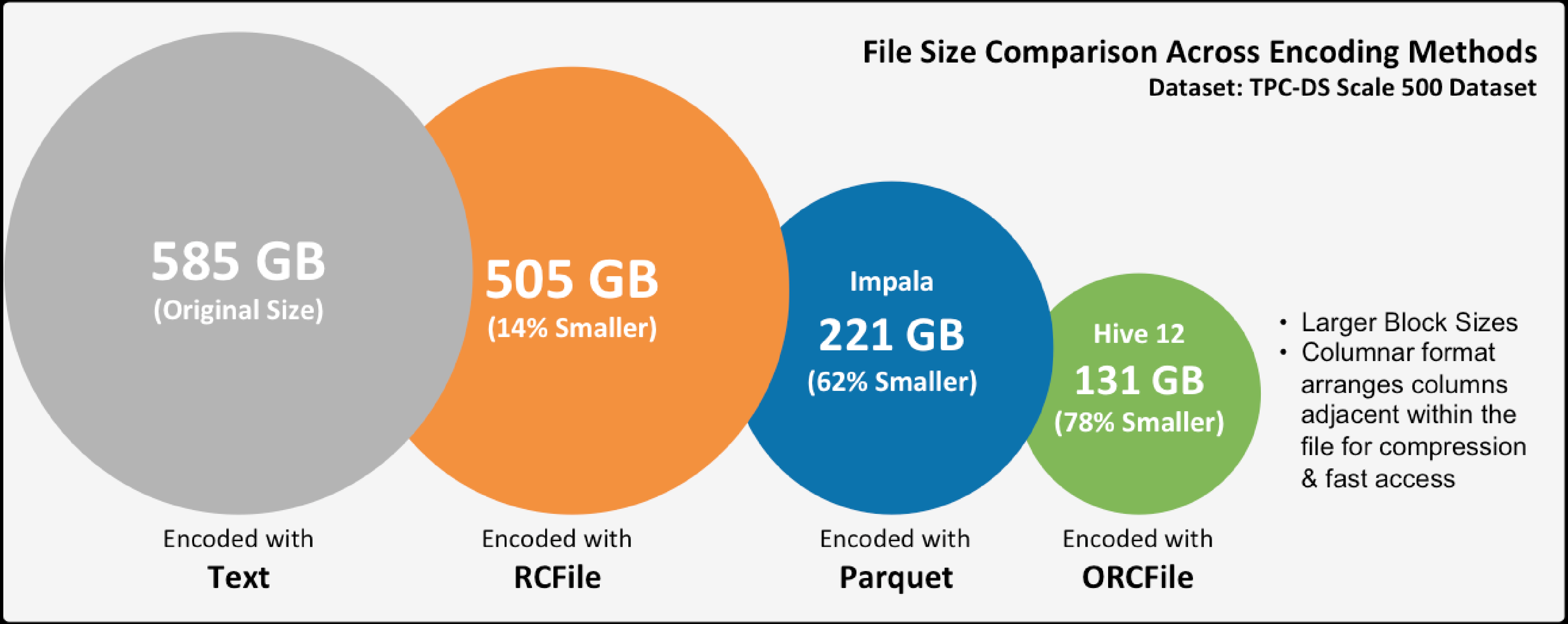
```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
[db_name.]table_name
     [(col_name data_type [COMMENT col_comment], ...)]
     [COMMENT table_comment]
     [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
     [CLUSTERED BY (col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)]
INTO num_buckets BUCKETS]
     [
       [ROW FORMAT row_format]
       [STORED AS file_format]
       | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
     ]
     [LOCATION hdfs_path]
     [TBLPROPERTIES (property_name=property_value, ...)]
     [AS select_statement];
```

```
ROW FORMAT SERDE serde_name
        [WITH SERDEPROPERTIES (prop_name=prop_value, ...)]
```

STORED AS – Storage file format can be specified in this clause. Below are the available file formats for hive table creation.

```
        SEQUENCEFILE
        TEXTFILE
        RCFILE
        PARQUET
        ORC
        AVRO
        INPUTFORMAT input_format_classname OUTPUTFORMAT
output_format_classname
```

We should not use LOAD DATA INPATH command to load data into any file format other than text file table when your source file is text. Always need to use INSERT INTO SELECT Clause only. While creation as STORED AS binaryformat is mutually exclusive with ROW FORMAT DELIMITED



- Larger Block Sizes
- Columnar format arranges columns adjacent within the file for compression & fast access

TBLPROPERTIES – Metadata key/value pairs can be tagged to the table. last_modified_user and last_modified_time properties are automatically added under table properties and managed by Hive. Some example predefined table properties are,

TBLPROPERTIES ("comment"="table_comment")
TBLPROPERTIES ("hbase.table.name"="table_name") //for hbase integration
TBLPROPERTIES ("immutable"="true") or ("immutable"="false")
TBLPROPERTIES ("orc.compress"="ZLIB") or ("orc.compress"="SNAPPY") or ("orc.compress"="NONE")
TBLPROPERTIES ("transactional"="true") or ("transactional"="false") default is "false"
TBLPROPERTIES ("NO_AUTO_COMPACTION"="true") or ("NO_AUTO_COMPACTION"="false"), the default is "false"

In Hive 0.14 onwards, Record level INSERT/DELETE/UPDATE are possible technically but it has lots of limitations and complexities behind the scenes which made it like it is not usable.

- For Every INSERT INTO statement, it runs a separate MR job and creates a small file
- For Update Statement it expects exclusive locks and locks not fully matured or reliable in Hive/Zookeeper setup.

- We do not recommend to enable transactional nature in Hive, better integrate with Hbase for the same

```
WITH T AS
 (SELECT ddf_id,
  ddf_ddf_id1,
  ddf_ddf_id2
 FROM nrsp_com.mrag_dde_formula
 WHERE ddf_id IN
  (SELECT DDO_OUF_ID
  FROM TEST.TMPO_DDE_OUTPUT_FACTS,
  TEST.TMPO_DDE_SETUP
  WHERE DDO_DDS_ID = DDS_ID
  AND DDS_ORD_ID = 93038)
  )
SELECT t1.*, t.ddf_id FROM t join TEST.trag_output_fact t1 on t.ddf_id = t1.ouf_id
Union all
SELECT t1.*, t.ddf_id FROM t join TEST.trag_output_fact t1 on t.ddf_ddf_id1 =
t1.ouf_id
Union all
SELECT t1.*, t.ddf_id FROM t join TEST.trag_output_fact t1 on t.ddf_ddf_id2 =
t1.ouf_id;
```

Sample Tables Creation

Sample Data for below tables → Download Here

```
DROP TABLE IF EXISTS user;

CREATE TABLE IF NOT EXISTS user (
first_name VARCHAR(64),
last_name VARCHAR(64),
company_name VARCHAR(64),
address STRUCT<zip:INT, street:STRING>,
country VARCHAR(64),
city VARCHAR(32),
state VARCHAR(32),
post INT,
phone_nos ARRAY<STRING>,
mail MAP<STRING, STRING>,
web_address VARCHAR(64)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '\t'
MAP KEYS TERMINATED BY ':'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/user/User_Records.txt' OVERWRITE INTO TABLE user;
```

Creating Table from other Table

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_db.user
LIKE default.user
LOCATION '/user/hive/usertable';

INSERT OVERWRITE TABLE test_db.user SELECT * FROM default.user;

SELECT first_name, city, mail FROM test_db.user WHERE country='AU';
```

Table with ORC File Format & Compression

```
STORED AS ORC
LOCATION '/user/hive/orc/user'
  TBLPROPERTIES ("orc.compress"="SNAPPY");
```

Views

```
Create view v1 as SELECT clause;
Drop view v1;
Describe v1;
```

Sample Data & Table Creation

```
$ hive
Hive history
file=/tmp/hadoop/hive_job_log_hadoop_201208022144_2014345460.txt
hive> !cat data/user-posts.txt;
user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394
hive>CREATE TABLE posts (user STRING, post STRING, time BIGINT)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
hive> show tables;
OK
posts
hive> describe posts;
user string
post string
time bigint
```

Load Data Into a Table

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'
> OVERWRITE INTO TABLE posts;
Copying data from file:/home/hadoop/Training/play_area/data/user-posts.txt
Copying file: file:/home/hadoop/Training/play_area/data/user-posts.txt
Loading data to table default.posts
Deleted /user/hive/warehouse/posts
OK
Time taken: 5.818 seconds
hive>dfs -cat /user/hive/warehouse/posts/user-posts.txt
user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394
```

Load Data Into a Table

```
hive> select count (1) from posts;
Total MapReduce jobs = 1
Launching Job 1 out of 1
...
Starting Job = job_1343957512459_0004, Tracking URL =
http://localhost:8088/proxy/application_1343957512459_0004/
Kill Command = hadoop job -Dmapred.job.tracker=localhost:10040 -kill
job_1343957512459_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2012-08-02 22:37:24,962 Stage-1 map = 0%, reduce = 0%
2012-08-02 22:37:30,497 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-08-02 22:37:32,664 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.64 sec
MapReduce Total cumulative CPU time: 2 seconds 640 msec
Ended Job = job_1343957512459_0004
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.64 sec HDFS Read: 0 HDFS Write: 0
SUCESS
Total MapReduce CPU Time Spent: 2 seconds 640 msec
OK
4
Time taken: 14.204 seconds
```

Query Data

```
hive> select count (1) from posts;
Total MapReduce jobs = 1
Launching Job 1 out of 1
...
Starting Job = job_1343957512459_0004, Tracking URL =
http://localhost:8088/proxy/application_1343957512459_0004/
Kill Command = hadoop job -Dmapred.job.tracker=localhost:10040 -kill
job_1343957512459_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2012-08-02 22:37:24,962 Stage-1 map = 0%, reduce = 0%
2012-08-02 22:37:30,497 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-08-02 22:37:32,664 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.64 sec
MapReduce Total cumulative CPU time: 2 seconds 640 msec
Ended Job = job_1343957512459_0004
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.64 sec HDFS Read: 0 HDFS Write: 0
SUCESS
Total MapReduce CPU Time Spent: 2 seconds 640 msec
OK
4
Time taken: 14.204 seconds
```

Query Data

```
hive> select * from posts where user="user2";
...
...
OK
user2 Cool Deal 1343182133839
Time taken: 12.184 seconds
hive> select * from posts where time<=1343182133839 limit 2;
...
...
OK
user1 Funny Story 1343182026191
user2 Cool Deal 1343182133839
Time taken: 12.003 seconds
hive>
```

Drop Table

```
hive> DROP TABLE posts;
OK
Time taken: 2.182 seconds
hive> exit;
$ hdfs dfs -ls /user/hive/warehouse/
```

Schema Violations

What would happen if we try to insert data that does not comply with the pre-defined schema?

```
hive> !cat data/user-posts-inconsistentFormat.txt;
user1,Funny Story,1343182026191
user2,Cool Deal,2012-01-05
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394

hive> describe posts;
OK
user string
post string
time bigint
Time taken: 0.289 seconds
```

Schema Violations

```
hive> LOAD DATA LOCAL INPATH
> 'data/user-posts-inconsistentFormat.txt'
> OVERWRITE INTO TABLE posts;
OK
Time taken: 0.612 seconds
hive> select * from posts;
OK
user1 Funny Story 1343182026191
user2 Cool Deal NULL
user4 Interesting Post 1343182154633
user5 Yet Another Blog 13431839394
Time taken: 0.136 seconds
hive>
```

Hive Built-In Functions

Mathematical Functions

- round
- floor
- ceil
- abs
- rand

Collection Functions

- size(Map) or size(Array)
- map_keys(Map)
- map_values(Map)
- array_contains(Array, value)
- sort_array(Array)

http://hadooptutorial.info/hive-functions-examples/

String Functions

- concat(‘foo’, ‘bar’)
- instr(string str, string substr) → Returns the position of the first occurence of substr in str
- length(string A)
- regexp_extract(string subject, string pattern, int index)
- split(string str, string pat)
- substr(string|binary A, int start, int len)
- translate(string input, string from, string to)

Aggregate Functions

- count(*) – Returns total no of rows
- count(DISTINCT col1) -- Distinct values
- sum(col)
- avg(col)
- min(col)
- max(col)

Hive CLI Commands

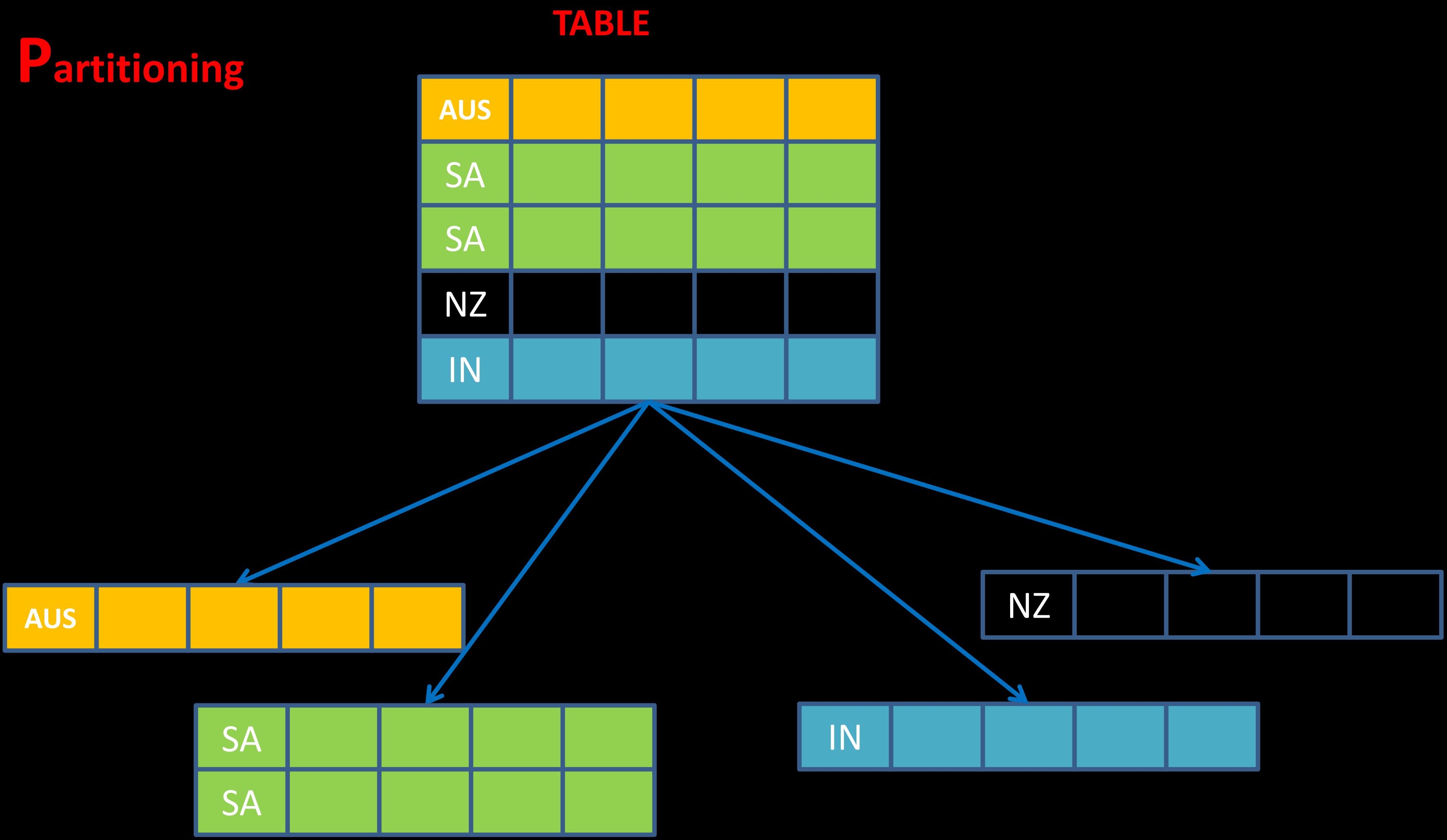
Argument	Description
-d,–define <key=value>	Defining new variables for Hive Session.
–database <databasename>	Specify the database to use in Hive Session
-e <quoted-query-string>	Running a Hive Query from the command line.
-f <filename>	Execute the hive queries from file
-h <hostname>	Connecting to Hive Server on remote host
-p <port>	Connecting to Hive Server on port number
–hiveconf <property=value>	Setting Configuration Property for current Hive Session
–hivevar <key=value>	Same as –define argument
-i <filename>	Initialization of Hive Session from an SQL properties file
-S,–silent	Silent mode in interactive shell, suppresses log messages
-v,–verbose	Verbose mode (prints executed SQL to the console)

For Examples refer http://hadooptutorial.info/hive-cli-commands/

Hive CLI Commands

Command	Description
quit or exit	Use quit or exit to leave the interactive shell.
set key=value	Set value of a configuration property/variable.
set	This will print all configuration variables if used without a property argument.
set -v	This will print all hadoop and hive configuration variables. Same as Set Command without arg.
reset	This will reset all the configuration properties to default, even if we provide any property argument.
add FILE[S] <file> <file>* add JAR[S] <file> <file>* add ARCHIVE[S] <file> <file>*	Adds a file(s)/jar(s)/archives to the hive distributed cache.
list FILE[S]	list all the files added to the distributed cache.
delete FILE[S] <file>*	Removes the resource(s) from the distributed cache.
! <cmd>	Executes a shell command from the hive shell
dfs	Executes a dfs command from the hive shell
<query>	Executes a hive query and prints results to standard out
source FILE <file>	Used to execute a script file inside the CLI.

Partitioning



- To increase performance Hive has the capability to partition data
- The values of partitioned column divide a table into segments
- Partitions are defined at the time of table creation using the PARTITIONED BY clause, with a list of column definitions for partitioning
- For example, In a large user table where the table is partitioned by country, then selecting users of country ‘IN’ will just scan one directory ‘country=IN’ instead of all the directories.

- Sample Data → Download Here

```
CREATE TABLE partitioned_user(
firstname VARCHAR(64),
lastname  VARCHAR(64),
address   STRING,
city   VARCHAR(64),
post      STRING,
phone1    VARCHAR(64),
phone2    STRING,
email     STRING,
web       STRING)
PARTITIONED BY (country VARCHAR(64), state VARCHAR(64))
STORED AS ORC;
```

Static Partitioning

```
hive> LOAD DATA LOCAL INPATH '${env:HOME}/staticinput.txt'
      INTO TABLE partitioned_user
      PARTITION (country = 'US', state = 'CA');
```

Table Directory Structure

```
/user/hive/warehouse/partitioned_user/country=US/state=CA/
                                      country=UK/state=LN/
                                      country=IN/state=AP/
                                      country=AU/state=ML/
```

Loading Partitions From Other Table & External Table Partitions

```
hive> INSERT OVERWRITE TABLE partitioned_user
      PARTITION (country = 'US', state = 'AL')
      SELECT fname,lname,addr,city,post,ph1,ph2,email,web FROM another_user au
      WHERE au.country = 'US' AND au.state = 'AL';
```

```
hive> ALTER TABLE partitioned_user ADD PARTITION (country = 'US', state = 'CA') LOCATION
'/hive/external/tables/user/country=us/state=ca'
```

http://hadooptutorial.info/partitioning-in-hive/

Show Partitions

```
 hive> SHOW PARTITIONS partitioned_user;
OK
country=AU/state=AC
country=AU/state=NS
country=AU/state=NT
```

Describe Partitions

hive> DESCRIBE FORMATTED partitioned_user PARTITION(country='US', state='CA');

Alter Partitions

```
ALTER TABLE partitioned_user ADD IF NOT EXISTS
PARTITION (country = 'US', state = 'XY') LOCATION '/hdfs/external/file/path1'
PARTITION (country = 'CA', state = 'YZ') LOCATION '/hdfs/external/file/path2'

ALTER TABLE partitioned_user PARTITION (country='US', state='CA')
SET LOCATION '/hdfs/partition/newpath';

ALTER TABLE partitioned_user DROP IF EXISTS PARTITION(country='US', state='CA');
ALTER TABLE partitioned_user PARTITION(country='US', state='CA') RENAME PARTITION TO
(country='US', state='TX');
```

Dynamic Partitioning

- Instead of loading each partition separately, which will result in writing lot of HQL statements for huge no of partitions, Hive supports dynamic partitioning with which we can add any number of partitions with single HQL execution.
- Hive will automatically splits our data into separate partition files based on the values of partition keys present in the input files.
- It gives the advantages of easy coding and no need of manual identification of partitions

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> INSERT INTO TABLE partitioned_user
        PARTITION (country, state)  SELECT  firstname ,
lastname  ,
address   ,
city      ,
post      ,
phone1    ,
phone2    ,
email     ,
web       ,
country   ,
state
FROM temp_user;
```

Bucketing

- ❑ Mechanism to query and examine random samples of data
- ❑ Break data into a set of buckets based on a hash function of a "bucket column"
- ❑ Capability to execute queries on a sub-set of random data
- ❑ Hive Doesn’t automatically enforce bucketing
- ❑ User is required to specify the number of buckets by setting # of reducer

```
hive> mapred.reduce.tasks = 32;
hive> hive.enforce.bucketing = true;
hive> CREATE TABLE post_count (user STRING, count INT)  CLUSTERED BY (user) SORTED BY
(user) INTO 5 BUCKETS;

hive> insert overwrite table post_count select user, count(post) from posts group by user;

hive> dfs -ls -R /user/hive/warehouse/post_count/
/user/hive/warehouse/post_count/000000_0
/user/hive/warehouse/post_count/000001_0
/user/hive/warehouse/post_count/000002_0
/user/hive/warehouse/post_count/000003_0
/user/hive/warehouse/post_count/000004_0

hive> select * from post_count TABLESAMPLE(BUCKET 1 OUT OF 2);
user1 2
user5 1
```

http://hadooptutorial.info/bucketing-in-hive/

Hive UDFs

- ❑ Regular UDFs (User defined functions)
- ❑ UDAFs (User-defined aggregate functions)
- ❑ UDTFs (User-defined table-generating functions).

Any custom UDFs that we are going to write must satisfy the following two properties:

- ❖ Must extend class org.apache.hadoop.hive.ql.exec.UDF .
- ❖ Must implement at least one evaluate() method.

- ➢ hive> ADD JAR /home/siva/AutoIncrementUDF.jar;
- ➢ hive> CREATE TEMPORARY FUNCTION incr AS 'AutoIncrementUDF';
- ➢ INSERT OVERWRITE TABLE increment_table1 SELECT incr() AS inc, id, c1, c2 FROM t1;

http://hadooptutorial.info/writing-custom-udf-in-hive-auto-increment-column-hive/

Hive JDBC Client

```
package com.test.hiveclient;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class HiveJdbcClientExample {
/*
* Before Running this example we should start thrift server. To Start
* Thrift server we should run below command in terminal hive --service hiveserver &
*/
private static String driverName = "org.apache.hive.jdbc.HiveDriver";

public static void main(String[] args) throws SQLException {
try {
Class.forName(driverName);
} catch (ClassNotFoundException e) {
e.printStackTrace();
```

Hive JDBC Client

```
System.exit(1);
}
Connection con =
DriverManager.getConnection("jdbc:hive2://quickstart.cloudera:10000/default", “hive",
“cloudera");
Statement stmt = con.createStatement();

String tableName = "empdata";
stmt.execute("drop table " + tableName);
ResultSet res = stmt.execute("create table " + tableName
+ " (id int, name string, dept string)");

// show tables
String sql = "show tables '" + tableName + "'";
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
if (res.next()) {
System.out.println(res.getString(1));
}
   // describe table
   sql = "describe " + tableName;
```

Hive JDBC Client

```
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(res.getString(1) + "\t" + res.getString(2) + "\t" + res.getString(2));
    }
    // load data into table
    String filepath = "/home/user/input.txt";
    sql = "load data local inpath '" + filepath + "' into table " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
sql = "select * from empdata where id='1'";
res = stmt.executeQuery(sql);
// show tables
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
System.out.println(res.getString(1));
System.out.println(res.getString(2));
System.out.println(res.getString(3));}
res.close(); stmt.close(); con.close(); }
}
```

HiveServer2 & Beeline

http://hadooptutorial.info/hiveserver2-beeline-introduction/

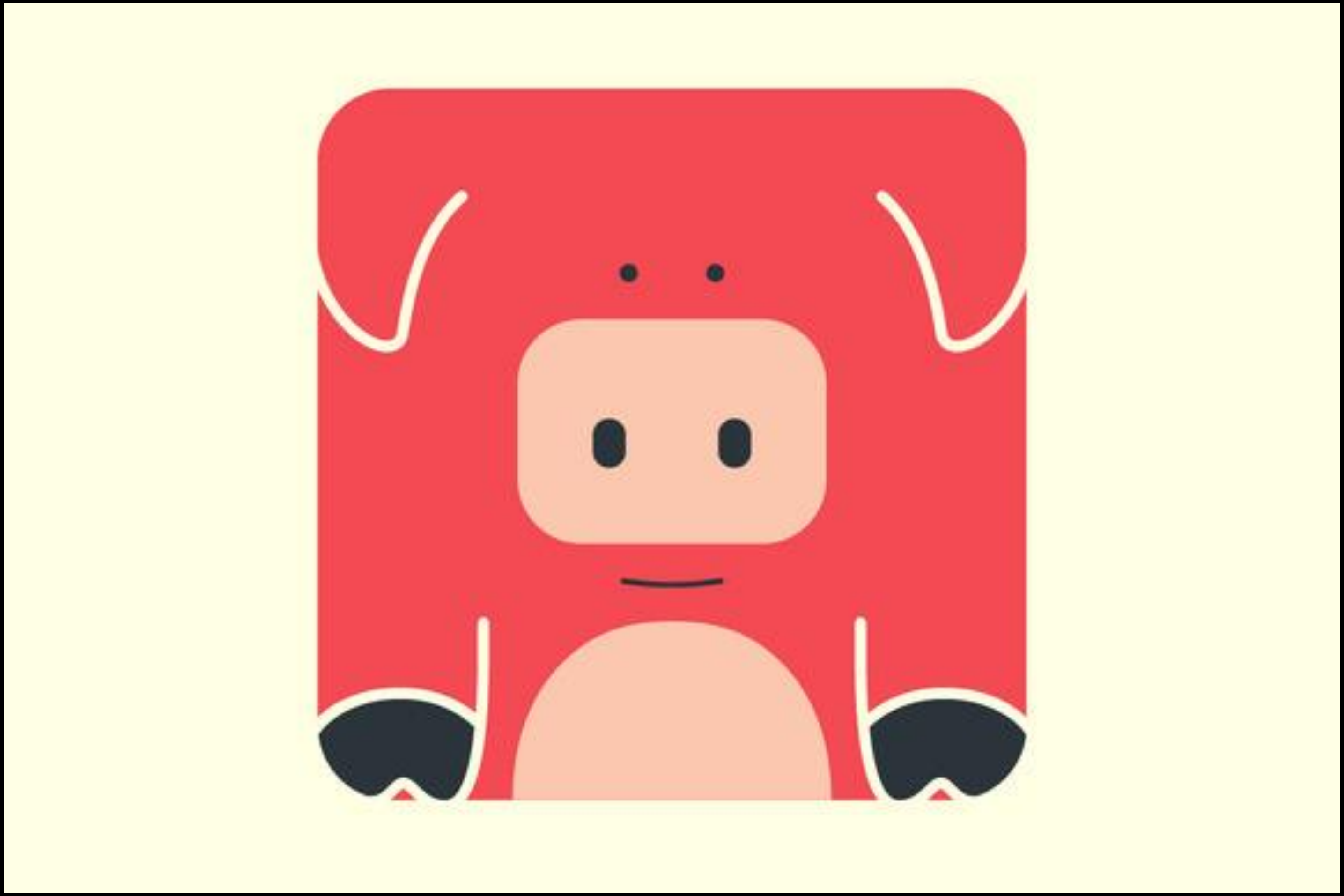
Hive Integration With Tools

- http://hadooptutorial.info/hbase-integration-with-hive/
- http://hadooptutorial.info/hive-on-tez/
- http://hadooptutorial.info/tableau-integration-with-hadoop/

Hive Performance Tuning

- http://hadooptutorial.info/hive-performance-tuning/

ANY QUESTIONS?



Thank you...