UNIT IV

HIVE

HIVE

- Introduction to Hive
- Hive Architecture
- Hive Data Types
- Hive File Format
- Hive Query Language (HQL)
- User-Defined Function (UDF) in Hive.

HIVE

- Hive is a Data Warehousing tool.
- Developed by Facebook to manage their ever-growing volumes of log data in the year 2007.
- In 2008 it became Apache Hadoop sub-project.
- Hive is used to query structured data built on top of Hadoop.
- It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.
- Hive makes use of the following:
  - HDFS FOR storage
  - MapReduce for execution
  - Stores metadata in an RDBMS

HIVE

- HIVE is a data warehousing tool.
- Hive suitable for Data warehousing applications.
- Hive processes batch jobs on huge data that is immutable.
- Web logs, Applications logs.

Recent Releases of HIVE

- Hive 0.10
  - Batch
  - Read-Only Data
  - HiveQL
  - MR

- Hive 0.13
  - Interactive
  - Read-Only Data
  - Substantial SQL
  - MR, TEZ

- Hive 0.14
  - Transactions with ACID semantics
  - Cost based optimizer
  - SQL temporary tables.

Features of Hive

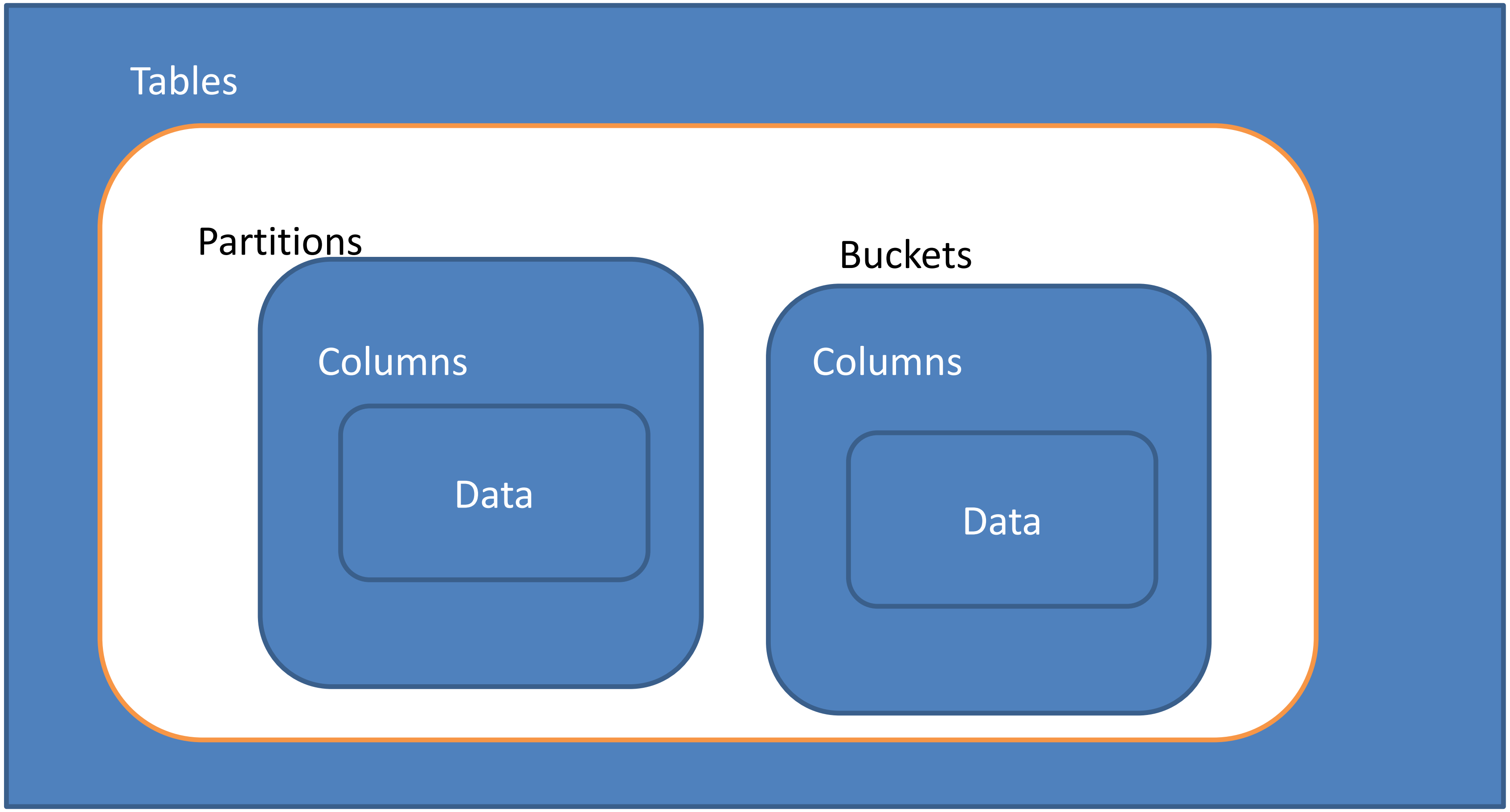
–It is similar to SQL.
–It provides SQL type language for querying called HiveQL or HQL, which is easy to code.
–Hive supports rich data types such as structs, lists, and maps.
–Hive suppots SQL filters, group-by and order-by clauses.
–Custom types, custom functions can be defined.

Hive Data Units

- Databases : The namespace for tables
- Tables: Set of records that have similar schema.
- Partitions: Logical separations of data based on classification of given information as per specific attributes.
- Buckets or clusters : Similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.

Data units arranged in HIVE

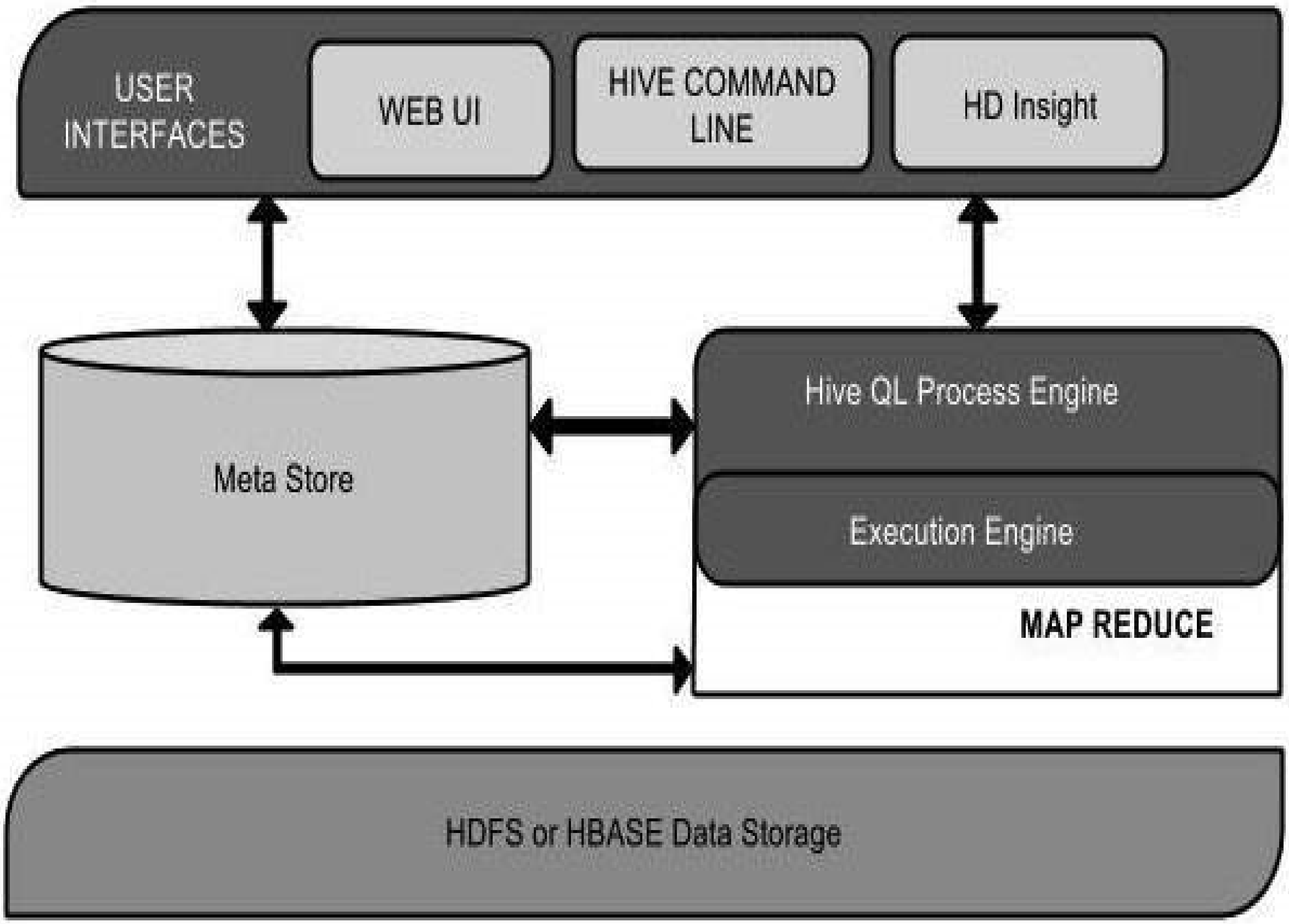
- Databases

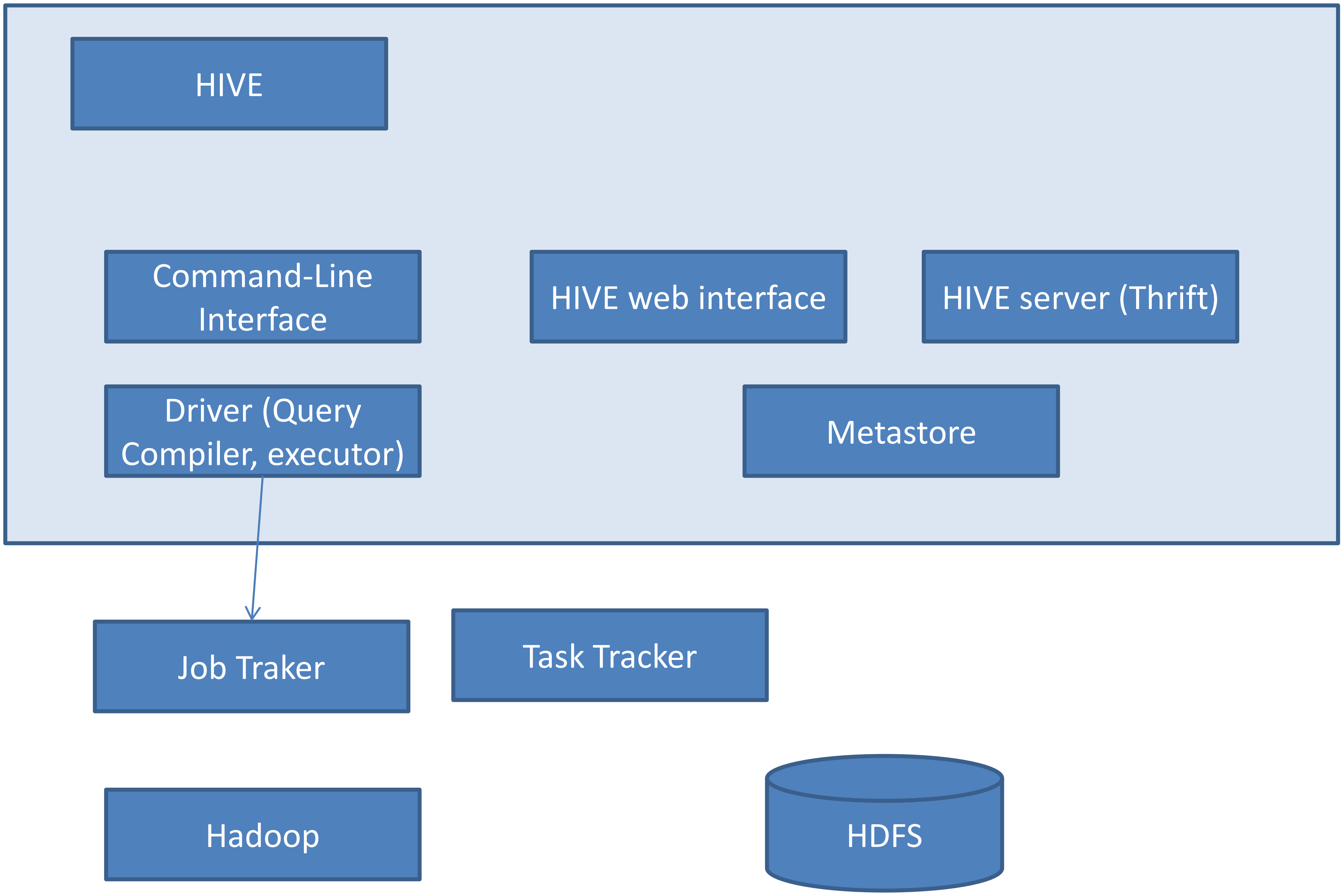


Hive data units

- A database contains several tables.
- Each table is constituted of rows and columns.
- In Hive tales are stored as a folder
- Partition tables are stored as sub-directory.
- Bucketed tables are stored as a file.

HIVE Architecture





HIVE Architecture

User Interface

- Hive is a data warehouse infrastructure software that can create interaction between user and HDFS.
- The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
- HIVE command-Line Interface (Hive CLI): The most commonly used interface to interact with HIVE.
- Hive Web Interface: It is a simple Graphic user Interface to interact with Hive and to execute

Meta Store

- Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
- Meta store consists of the following:
  - Metastore service: Offers interface to the Hive.
  - Database: Stores data definitions, mappings to the data and others.

HiveQL Process Engine

- HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.

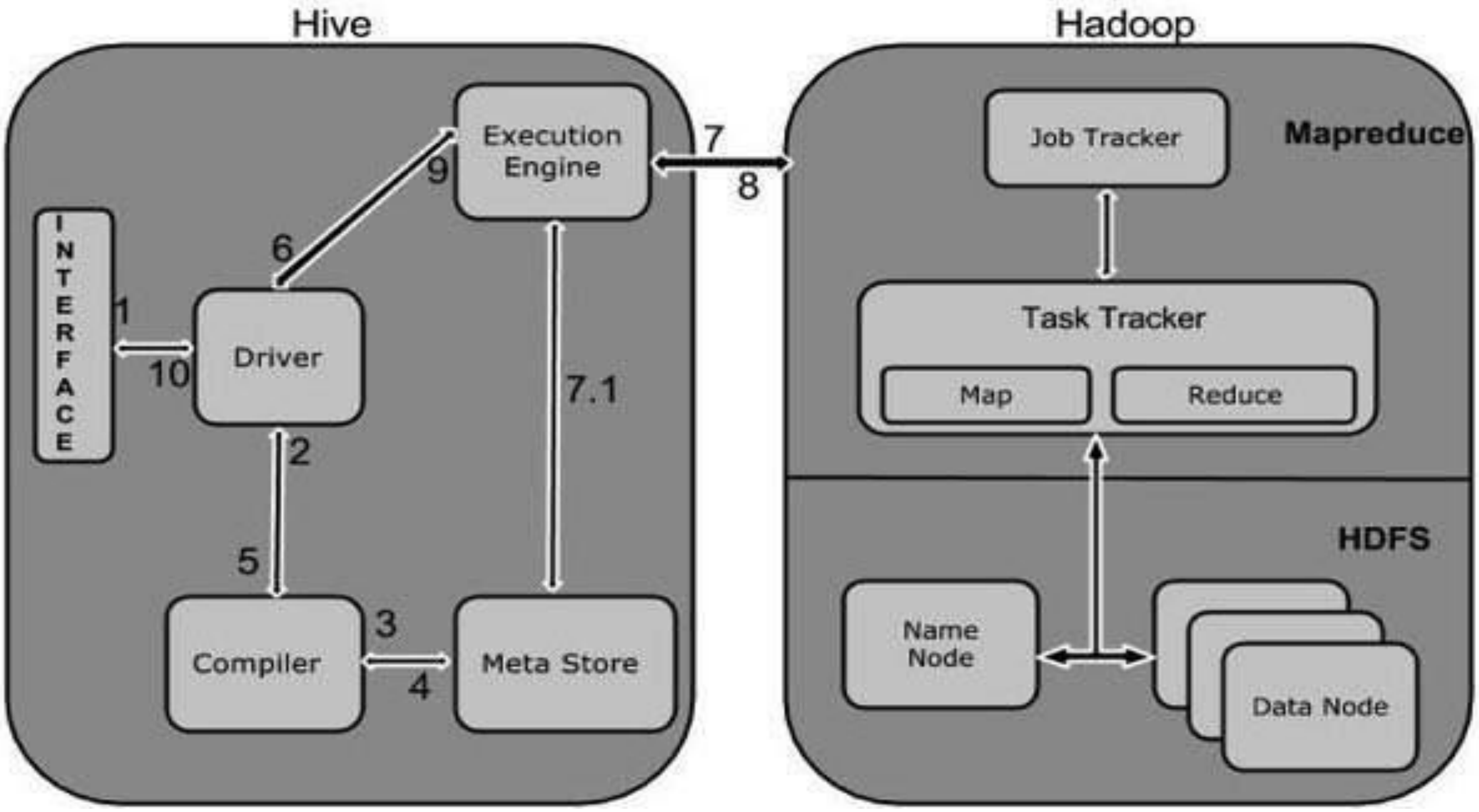
Execution Engine

- The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the MapReduce.

HDFS or HBASE

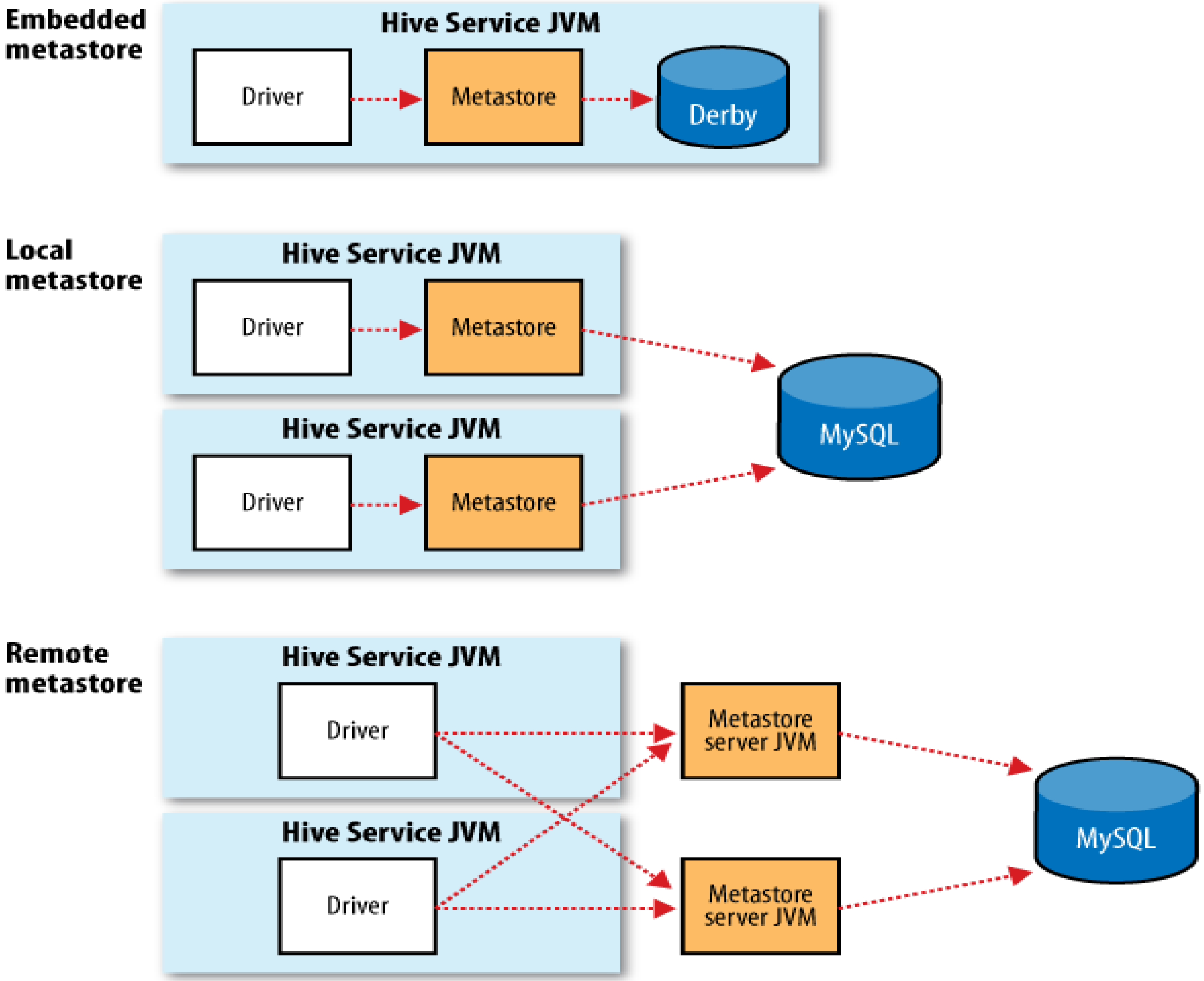
- Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working of Hive



Metastore

- There are three kinds of metastore.
- Embedded Metastore
- Local Metastore
- Remote Metastore


Embedded metastore
Hive Service JVM
Driver
Metastore
Derby
Local metastore
Hive Service JVM
Driver
Metastore
Hive Service JVM
Driver
Metastore
MySQL
Remote metastore
Hive Service JVM
Driver
Hive Service JVM
Driver
Metastore server JVM
Metastore server JVM
MySQL

Embedded Metastore

This metastore is mainly used for unit tests.

Here one process is allowed to connect to the metastore at a time.

This is the default metastore for HIVE.

It is Apache Derby Hive Server Process.

Local Metastore Metastore

Metadata can be stored in any RDBMS component like MySQL.
Local Metastore allows multiple connections at a time.
In this mode, the Hive metastore service runs in the main Hive Server process,
but he metastore database runs in a separate process on separate host.

Remote Metastore

The Hive driver and the metastore interface run on different JVMs.
The databased can be fire-walled from the Hive User and also database credentials are completely isolated from the users of Hive.

Hive Data Types

- All the data types in Hive are classified into four types, given as follows:
- Column Types
- Literals
- Null Values
- Complex Types
- Column Types
- Column type are used as column data types of Hive. They are as follows:
- Integral Types
- String Types
- Dates
- Decimals
- Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.
- The following table depicts various INT data types:
- Type Postfix Example
- TINYINT Y 10Y
- SMALLINTS 10S
- INT - 10
- BIGINT L 10L

•String Types

•String type data types can be specified using single quotes (' ') or double quotes (" ").
• It contains two data types: VARCHAR and CHAR.
•Hive follows C-types escape characters.
•The following table depicts various CHAR data types:

•Data Type	Length
•VARCHAR	1 to 65355
•CHAR	255

•Timestamp
•It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” and format “yyyy-mm-dd hh:mm:ss.ffffffffff”.

- Dates
- DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

- Decimals
- The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:
- DECIMAL(precision, scale) decimal(10,0)
- Union Types
- Union is a collection of heterogeneous data types. You can create an instance using **create union**. The syntax and example is as follows:

Complex Types

- The Hive complex data types are as follows:
- Arrays
- Arrays in Hive are used the same way they are used in Java.
- Syntax: ARRAY<data_type>Maps
- Maps in Hive are similar to Java Maps.
- Syntax: MAP<primitive_type, data_type>Structs
- Structs in Hive is similar to using complex data with comment.

HIVE FILE FORMAT

- Text File
  - Each record is a line in the file.
  - Different control characters are used as delimiters.
  - CSV, TSV , JSON or XML
- Sequential File
  - Sequential files are flat files that store binary key-value pairs. It includes compression.
- RCFile (Record Columnar File)

RC File (Record Columnar File)

- RC File stores the data in Column oriented manner which ensures that aggregation operation is not an expensive operation.
- RC file first partitions horizontally and then vertically to serialize the data.

•C1	c2	c3	c4
•11	12	13	14
•21	22	23	24
•31	32	33	34
•41	42	43	44
•51	52	53	54

- Table with two row groups

C1	c2	c3	c4	c1	c2	c3	c4
11	12	13	14	41	42	43	44
21	22	23	24	51	52	53	54
31	32	33	34				

- Table in RCFile format

Rowgroup 1	row group 2
11,21,31;	41,51;
12,22,32;	42,52;
13,23,33;	43,53;
14,24,34;	44,54;

HIVE Query Language (HQL)

- Create and manage tables and partitions
- Support various Relational , arithmetic and logical operators.
- Evaluate functions
- Download the content of a table to a local directory or result of queries to HDFS directory.

Data Definition Language(DDL)

- These statements are used to build and modify the tables and other objects.
- Create/Drop/alter database
- Create/drop/truncate table
- Alter table/partition/column
- Create/drop/alter view
- Create/drop/alter index
- Show
- describe

DML Data Manipulation statements

- To retrieve, store, modify, delete and update the data in database.
- The DML commands are as follows:
  - Loading files into table.
  - Inserting data into hive tables from queries.

Databases

- To create a database names students with comments and database properties.
- Create database if not exists students comment ‘student details’ with dbproperties (‘creator’ = ‘john’);
- Create database student;
- It will create in the default location /user/hive/warehouse.
- To display a list of all databases
  - Show databases;
- To describe a database
  - Describe database students;
- To describe the extended database
  - Describe database extended students;
- To alter the database properties
  - Alter database students set dbproperties(‘edited-by’ = ‘james’);
- To make the database as current working database
  - Use students;

Tables

- Hive provides two kinds of tables:
  - Managed tables
  - External tables
- Managed tables
  - Hive stores the managed tables under the warehouse folder under Hive.
  - The complete life cycle is managed by Hive.
  - When the internal table is dropped, it drops the data as well as the metadata.

Create table

- CREATE TABLE IF NOT EXISTS STUDENT (rollno INT, name STRING, gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ’\t’;

```
DESCRIBE STUDENT;
Rollno        int
Name          string
Gpa           float
```

External or self-managed tables

- When the table is dropped, it retains the data in the underlying location.
- Create external table if not exists ext_student (rollno int, name string, gpa float) row format delimited fields terminated by “\t” location ‘/student’
- TO load data into table from file
  - –Load data local inpath ‘/root/file.tsv’ overwrite into table ext_student;
- LOCAL keyword is used to load the data from local file system.
- To load the data from HDFS remove the local

Query

```
Select * from ext_student;
Select rollno,name from student;
```

Collection Data Types

- CREATE TABLE STUDENT_INFO (rollno int, name string, marks MAP<string,int>) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY':' MAP KEYS TERMINATED BY '!';
- LOAD DATA LOCAL INPATH '/root/hivedemos/studentinfo.csv' INTO TABLE STUDENT_INFO;
- 1001,John,sub1!56:sub2!78:sub3!60

QUERYING TABLE WITH COLLECTION DATA TYPES

- SELECT * from STUDENT_INFO;
- SELECT name,marks['sub1'] from STUDENT_INFO;

USING ARRAYS

- CREATE TABLE STUDENT_INFO (rollno int, name string,friends ARRAY<STRING>,marks MAP<string,int>) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY':' MAP KEYS TERMINATED BY '!';
- LOAD DATA LOCAL INPATH '/root/hivedemos/studentinfo.csv' INTO TABLE STUDENT_INFO;
- 1001,John,JAMES:RAJU,sub1!56:sub2!78:sub3!60
- SELECT NAME,FRINDS[0] FROM STUDENT_INFO;

Partitions

•In Hive the query reads the entire dataset even though a where clause filter is specified on a particular column.
•This becomes a bottleneck in most of the MapReduce jobs as it involves huge degree of I/O.
To reduce I/O required by the MapReduce job to improve the performance of the query.
•Partitions split the larger dataset into more meaningful chunks.

- Hive provides two kinds of partitions: static and dynamic partitions.
- Static : static partitions comprise columns whose values are known at compile time.
- CREATE TABLE IF NOT EXISTS STATIC_PART (ROLLNO INT, NAME STRING) PARTITIONED BY (GPA FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘\t’;

•Insert input data files individually into a partition table is Static Partition
•Usually when loading files (big files) into Hive tables static partitions are preferred
•Static Partition saves your time in loading data compared to dynamic partition You “statically” add a partition in table and move the file into the partition of the table.
•We can alter the partition in static partition
•You can get the partition column value form the filename, day of date etc without reading the whole big file.
•If you want to use Static partition in hive you should set property

•set hive.mapred.mode = strict

•This property set by default in hive-site.xml Static partition is in Strict Mode
•You should use where clause to use limit in static partition
•You can perform Static partition on Hive Manage table or external table.

TO LOAD DATA INTO PARTITION TABLE FROM TABLE

```
INSERT OVERWRITE TABLE STATIC_PART_STUDENT
PARTITION(GPA=4.0) SELECT ROLLNO,NAME FROM EXT_Student where
gpa=4.0;
```

```
ALTER TABLE STATIC_PART_Student add partition (gpa=3.5);
```

```
Insert overwrite table static_part_student partition (gpa=4.0) select rollno, name
from ext_student where gpa=4.0;
```

static partitioning

We need to specify the partition column value in each and every LOAD statement.

Suppose we are having partition on column country for table t1(userid, name,occupation, country), so each time we need to provide country value

```
hive>LOAD DATA INPATH '/hdfs path of the file' INTO TABLE t1
PARTITION(country="US")
```

```
hive>LOAD DATA INPATH '/hdfs path of the file' INTO TABLE t1
PARTITION(country="UK")
```

DYNAMIC PARTITIONS

Dynamic Partition have columns whose values are known only at Execution Time.
To create dynamic partition on column gpa.

```
CREATE TABLE IF NOT EXISTS DYNAMIC_PART_STUDENT (rollno int, name
string) partitioned by (gpa float) row format delimited fields terminated by ‘\t’;
```

To load data into dynamic partition table from table.

```
Set hive.exec.dynamic.partition=true;
Set hive.exec.dynamic.partition.mode=nonstrict;
```

```
INSERT OVERWRITE TABLE DYNAMIC_PART PARTITION (GPA) SELECT
rollno,name,gpa from ext_Student;
```

Dynamic Partition

It allow us not to specify partition column value each time.

The approach we follows is as below:

create a non-partitioned table t2 and insert data into it.

now create a table t1 partitioned on intended column(say country).

load data in t1 from t2 as below:

```
hive> INSERT INTO TABLE t2
PARTITION(country) SELECT * from T1;
```

make sure that partitioned column is always the last one in non partitioned table(as we are having country column in t2)

Bucketing

- In a partition, it will create a partition for each unique value of the column.
- This may lead to situations where you may end up with thousands of partitions.
- This can be avoided by using bucketing in which you can limit the number of buckets to create.
- A bucket is a file whereas a partition is a directory.
- Set hive.enforce.bucketing=true
- Crate table if not exists student (rollno int,name string, grade float) clustered by (grade) into 3 buckets;

LOAD DATA TO BUCKETED TABLE

```
FROM STUDENT INSERT OVERWRITE TABLE STUDENT_BUCKET SELECT
rollno, name, grade;
```

To display content of first bucket

```
SELECT DISTINCT GRADE FROM STUDENT_BUCKET
TABLESAMPLE(BUCKET 1 OUT OF 3 ON GRADE)
```

VIEWS

In HIVE VIEW support is available after 0.6. Views are purely logical object.

```
CREATE VIEW STUDENT_VIEW AS SELECT rollno,name from EXT_Student;

SELECT * FROM STUDENT_VIEW LIMIT 4;

DROP VIEW STUDENT_VIEW;
```

SUB-QUERY

In hive sub-queries are supported in the FROM CLAUSE. You need to specify name for sub-query because every table in a FROM clause has a name.

Write a subquery to count the occurrence of similar words in the file.

```
CREATE TABLE docs (line STRING);
Load data local inpath ‘/root/hivedemo/lines.txt’ OVERWRITE INTO TABLE docs;

CREATE TABLE word_count AS SELECT word, count(1) as count from (select
explode(split(line,’ ‘) ) AS word FROM docs) w GROUP BY WORD
ORDER BY WORD;

SELECT * FROM word_count;
```

joins

Joins in hive is similar to the SQL join.
To join student and department

```
CREATE TABLE IF NOT EXISTS STUDENT (rollno int, name string, gpa float)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘\t’;

LOAD DATA LOCAL INPATH ‘/ROOT/HIVEDEMOS/STUDENT.TSV’
OVERWRITE INTO TABLE STUDENT;

CREATE TABLE IF NOT EXISTS DEPARTMENT (rollno  int , deptno int, name
string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘\t’;

LOAD DATA LOCAL INPATH ‘/ROOT/HIVEDEMOS/DEPARTMENT.TSV’
OVERWRITE INTO TABLE DEPARTMENT;

Select a.rollno, a.name, a.gpa , b.deptno FROM STUDENT A JOIN
DEPARTMENT B ON A.ROLLNO=B.ROLLNO;
```

AGGREGATION

HIVE SUPPORTS aggregation functions like avg, count etc.

```
SELECT AVG(GPA) FROM STUDENT;
SELECT COUNT(*) FROM STUDENT;
```

GROUP BY AND HAVING

DATA in a column or columns can be grouped on the basis of values by using “group by”. Having clause is used to filter out groups NOT meeting the specified conditions.

```
SELECT rollno, name, gpa FROM STUDENT GROUP BY rollno,name,gpa
HAVING gpa>4.0;
```

RCFILE implementation

RC FILE (Record Columnar file) is a data placement structure that determines how to store relational tables on computer clusters.

```
CREATE TABLE STUDENT_RC (rollno int, name string, gpa float) STORED AS RCFILE;
INSERT OVERWRITE table STUDENT_RC SELECT * FROM STUDENT;

SELECT SUM(GPA) FROM STUDENT_RC;
```

SERDE STANDS FOR SERIALIZATION AND DESERIALIZATION
SERDE

1. CONTAINS the logic to convert unstructured data into records
2. Implemented using java
3. Serializers are used at the time of writing
4. Deserializers are used at query time (select statement)

Serializer takes a java object that hive has been working with and translates it into something that Hive can write into HDFS

Deserializer interface takes a binary representation or string of a record, converts it into a java object that Hive can then manipulate.

To manipulate the XML data

```
CREATE TABLE XMLSAMPLE(XMLDATA STRING);
LOAD DATA LOCAL INPATH ‘/ROOT/HIVEDEMOS/INPUT.XML’ INTO TABLE
XMLSAMPLE;

CREATE TABLE xpath_table as SELECT
XPATH_INT(XMLDATA,’EMPLOYEE/EMPID’), XPATH_STRING(XMLDATA,
‘employee/name’),
XPATH_STRING(XMLDATA, ‘employee/designation’)
From xmlsample;

Select * from xpath_table;
```

User Defined Functions

- We can use custom functions by defining the User-Defined Function (UDF)
- 1. Write java code
- Import org.apache.hadoop.hive.ql.exec.UDF;
- Convert java program into jar
- Then execute