Introduction to Hive
A.Baskar

What is Hive?

Hive is a Data Warehousing tool. Hive is used to query structured data built on top of Hadoop. Facebook created Hive component to manage their ever- growing volumes of data. Hive makes use of the following:

1.HDFS for Storage

1.MapReduce for execution

1.Stores metadata in an RDBMS.

Hive suitable for

1.Hive suitable for DWH applications.
2. processes batch jobs on huge data that is immutable.
3. EG: Web logs, Application logs.

Hive history

2007 : Born at FACEBOOK to analyse incoming data

2008 : Hive became Apache Hadoop sub-project.

Hive versions:

Hive 0.10

1.Batch
2.Read only data
3.HiveQL
4.MR

Hive 0.13

1.Interactive
2.Read-only data
3.Substantial SQL
4.MR

Hive 0.14

1.Transactions with ACID semantics
2.Cost based optimizer
3.SQL temporary tables.

Features of Hive

1.It is similar to SQL.

2.HQL is easy to code.

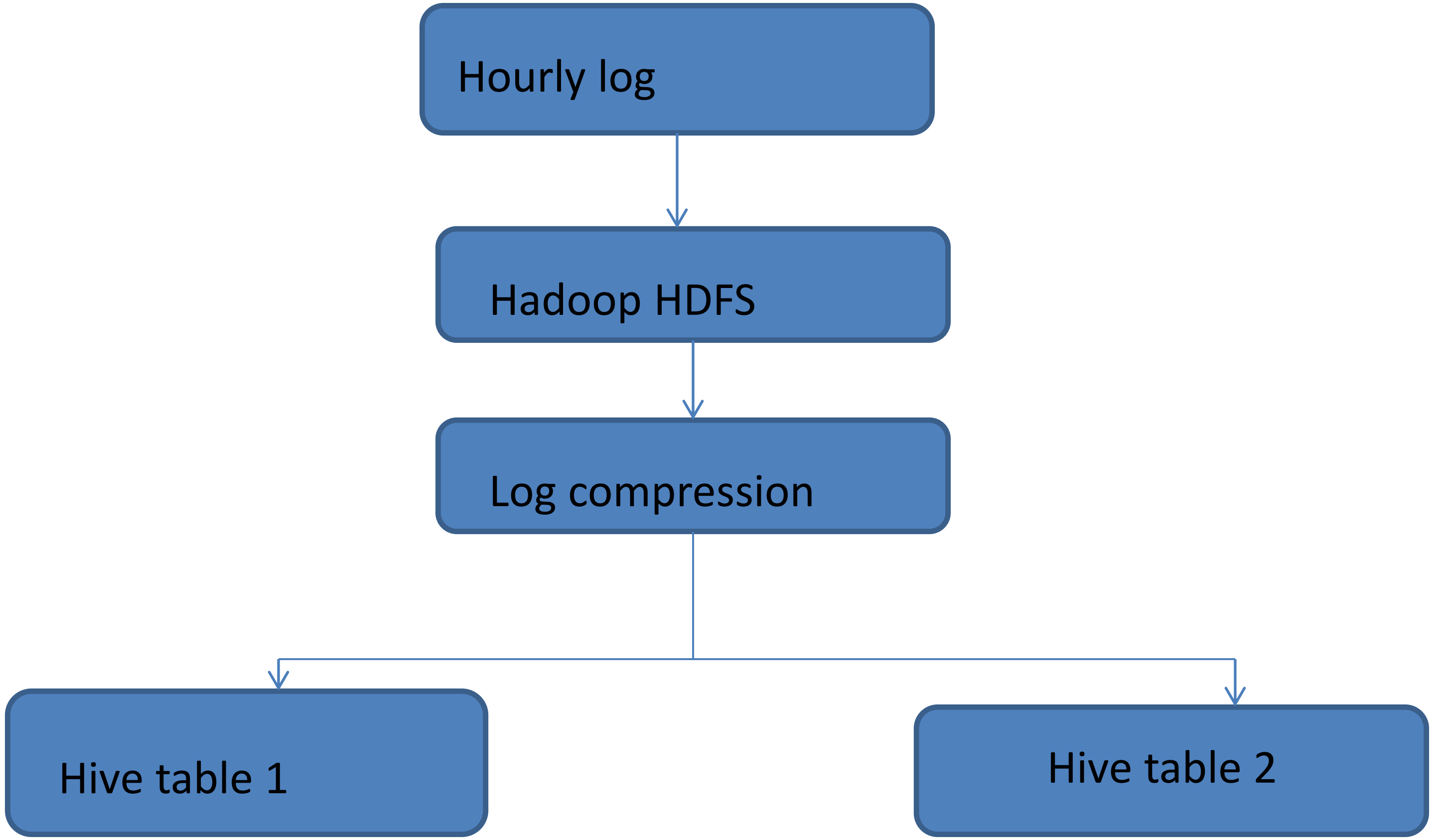
3.Hive supports rich data types such as structs, lists, and maps.

4.Hive supports SQL filters, group-by and order-by clauses.

5.Custom Types, Custom functions can be defined.

Hive integration and work flow

Work flow of log analysis file.



Hourly log

Hadoop HDFS

Log compression

Hive table 1

Hive table 2

Hive Data Units

Hive Data Units

- **Databases**: The namespace for tables

- **Tables** : Set of records that have similar schema

- **Partitions**: Logical separation of data based on classification of given information as per specific attributes. Once hive has partitioned the data based on specified key, it starts to assemble the records into specific folders as when the records are inserted.

- **Buckets (Clusters**) : Similar to partition but uses hash function to segregate the data and determines the cluster or bucket into which the record should be placed.

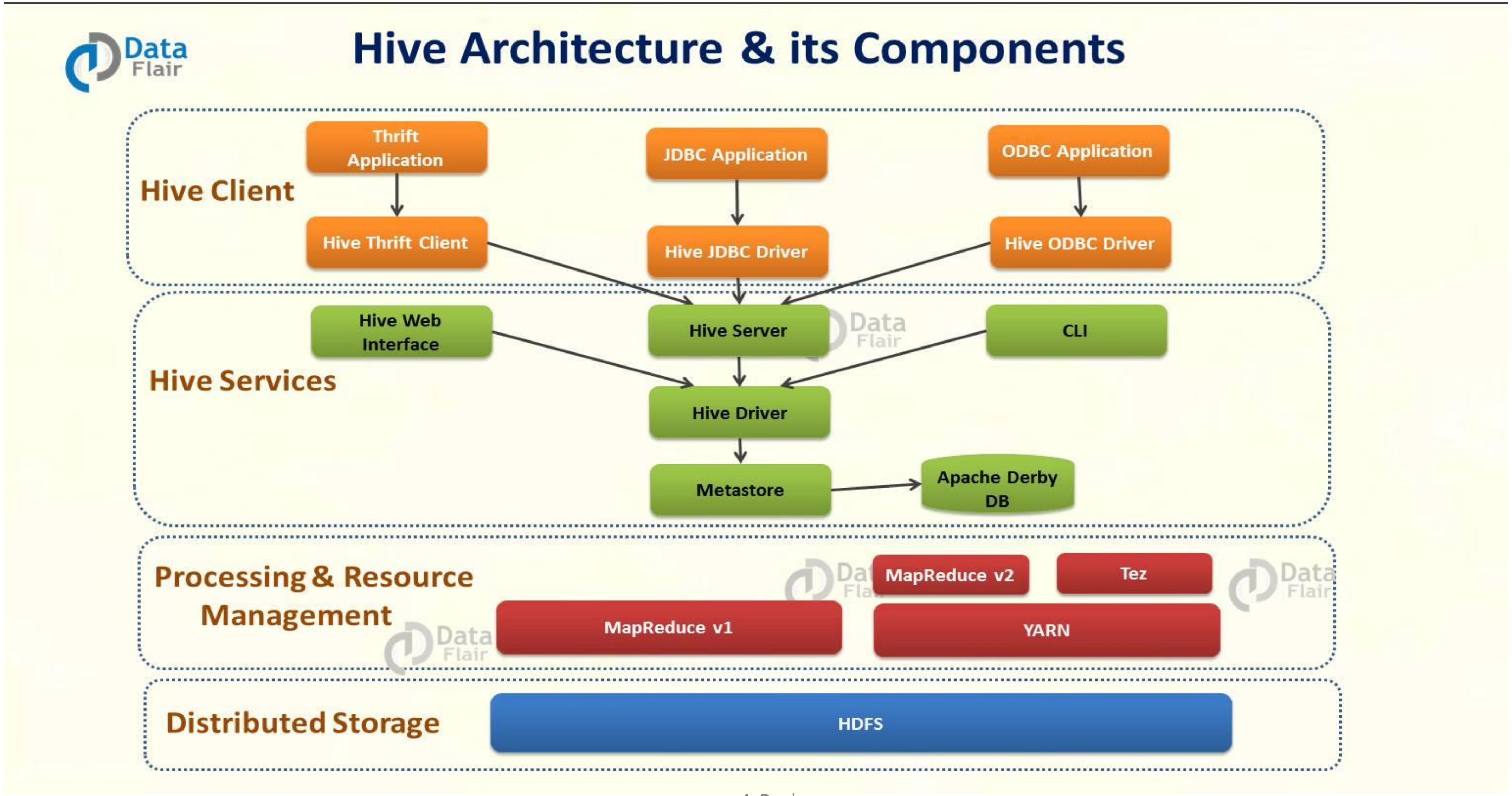
Semblance of hive structure with database

Database: Several tables, table constitute with rows and columns.

Hive:

Tables :	Folder
Partition tables:	sub directory
Bucketed tables:	stored as files.

Hive Architecture



Hive Meta store

Meta store : Hive table definitions and mappings to the data are stored in Meta store.

Meta store consists of the following:

Metastore services: Offers interface to the hive

Databases: Stores data definitions, mappings to the data and others.

Meta data includes ,

ID s of DBS

IDs of Tables

IDs of Indexes

The time of creation of tables

Input format

Output format

Types of Meta Sore

Embedded Meta store:

Used for unit tests
Default meta store
only one process is allowed to connect at a time
Here both DB and Meta store services embedded with main Hive server process.

Local meta store:

Meta data can be stored in any RDBMS components like MySQL.
Allows multiple connections at a time
Here Meta store service runs in Hive Server but DB runs in separate process.

Remote Meta store:

Hive driver and Meta store interface runs on different JVM.
This way the DB can be firewalled from the hive users and also DB credentials are completely isolated from hive users.

Hive Data Types

Numerical data type	
TINYINT	1- byte signed integer
SMALLINT	2- byte signed integer
INT	4- byte signed integer
BIGINT	8- byte signed integer
FLOAT	4-Byte single precision floating point
DOUBLE	8- Byte single precision floating point

String Type	Length
VARCHAR	1 to 65355
CHAR	255
Other Data types	
Boolean	
Binary	

Hive Data Types

- COLLECTION DATA TYPES
  - STRUCT
  - MAP
  - ARRAY

Hive File Format

- **Text File**

  The default file format is text file.

- **Sequential File**

  Sequential files are flat files that store binary key-value pairs.

- **RCFile (Record Columnar File)**

  RCFile stores the data in **Column Oriented Manner** which ensures that **Aggregation** operation is not an expensive operation.

Hive Query Language

Hive Query Language (HQL)

1.Create and manage tables and partitions.
2.Support various Relational, Arithmetic, and Logical Operators.
3.Evaluate functions.

4.Download the contents of a table to a local directory or result of queries to HDFS directory.

DDL and DML statements

Database

To create a database named “STUDENTS” with comments and database properties.

```
CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT 'STUDENT Details'  WITH DBPROPERTIES
('creator' = 'JOHN');
```

Database

To describe a database.

DESCRIBE DATABASE STUDENTS;

Shows only DB Name, comment, DB directory

DESCRIBE DATABASE EXTENDED STUDENTS;

Shows DB properties also.

Database

To display, a list of all databases

```
SHOW DATABASES;
```

To alter database properties

```
ALTER DATABASE STUDENTS SET DBPROPERTIES(‘Edited-by’=‘XX’);
```

To make the databases as current working database

```
USE STUDENTS;
```

Database

To drop database.

DROP DATABASE STUDENTS;

All databases stored in data warehouse directory in hive.

Tables

Hive provides two kinds of tables:

Managed Table

1. Hive stores the managed tables under the ware house folder under Hive
2. The complete life cycle of table and data is managed by Hive
3. When the internal table is dropped it drops the data as well as meta data.

External Table or self managed table

1. When the table is dropped , it retains the data in the underlying location
2. External Keyword is used.
3. Location needs to be specified to store the dataset in that particular location.

Tables

To create **managed table** named ‘STUDENT’.

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

Tables

To create **external table** named ‘EXT_STUDENT’.

```
CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name STRING,gpa FLOAT) ROW
FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION ‘/STUDENT_INFO';
```

Tables

To load data into the table from file named student.tsv.

```
LOAD DATA LOCAL INPATH ‘/root/hivedemos/student.tsv' OVERWRITE INTO
TABLE EXT_STUDENT;
```

Local keyword is used to load the table in local file systems.
To store in HDFS remove local keyword.

Querying Tables

To retrieve the student details from “EXT_STUDENT” table.

```
SELECT * from EXT_STUDENT;
```

Collection Data types in Tables

Map(Key- Value)

Input :

```
1001, John,Smith:Jones,Mark1!45:Mark2!67:Mark3!75
 1002, Jack,Smith:Jones,Mark1!55:Mark2!69:Mark3!85
```

```
CREATE TABLE STUDENT_INFO(rollno INT,name String,sub ARRAY<STRING>,marks MAP<STRING,INT>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’
COLLECTION ITEMS TERMINATED BY ‘:’
MAP KEYS TERMINATED BY ‘!’;
```

To load data

```
_LOAD DATA LOCAL INPATH ‘/root/hivedemos/studentinfo.csv’ INTO TABLE STUDENT_INFO
```

Querying collection Data types

```
SELCET * FROM STUDENT_INFO;

 1001   John  {Smith:Jones}   {“Mark1”:45,”Mark2”:67,”Mark3”:75}
 1002 Jack   {Smith:Jones}    {“Mark1”:55,”Mark2”:69,”Mark3”:85}
```

```
SELECT NAME,SUB FROM STUDENT-INFO;
 John  {Smith:Jones}
 Jack {Smith:Jones}
```

```
SELECT NAME,MARKS[‘MARKS1’] FROM STUDENT-INFO;
JOHN 45
JACK  55
```

```
SELECT NAME,SUB[0] FROM STUDENT-INFO;
John  Smith
Jack  Smith
```

Partitions

Partitions split the larger dataset into more meaningful chunks.
Hive provides two kinds of partitions: **Static Partition and Dynamic Partition.**

Static partition: comprise columns whose values are known at compile time.

Dynamic partition: Dynamic partition have columns whose v values are known only at Execution time.

Static partitions

- To create static partition based on “gpa” column.

```
CREATE TABLE IF NOT EXISTS STATIC_PART_STUDENT (rollno INT, name STRING) PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS  TERMINATED BY '\t';
```

- Load data and querying 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;
```

To add one more static partition based on gpa

```
ALTER TABLE STATIC-PART –STUDENT ADD PARTITION(GPA=3.5);
```

Partitions

- To create **dynamic partition** on column date.

```
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 a dynamic partition table from table.

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

Note: The dynamic partition strict mode requires at least one static partition column. To turn this off,
set hive.exec.dynamic.partition.mode=nonstrict

```
INSERT OVERWRITE TABLE DYNAMIC_PART_STUDENT PARTITION (gpa) SELECT
rollno,name,gpa from EXT_STUDENT;
```

Partition Example

Input File

id, name, dept, yoj
1, gopal, TP, 2012
2, kiran, HR, 2012
3, kaleel,SC, 2013
4, Prasanth, SC, 2013

_After partition

/tab1/employeedata/2012/file2

1, gopal, TP, 2012
2, kiran, HR, 2012

/tab1/employeedata/2013/file3

3, kaleel,SC, 2013
4, Prasanth, SC, 2013

Renaming a Partition

```
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION
partition_spec;
```

Eg:

```
 ALTER TABLE employee PARTITION (year=’1203’) RENAME TO PARTITION
(Yoj=’1203’);
```

Dropping a Partition

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION
partition_spec,...;
```

Eg:

```
ALTER TABLE employee DROP [IF EXISTS] PARTITION (year=’1203’);
```

Bucketing

Bucketing is similar to partition.

Difference between partitioning and Bucketing:

Partition need to create partition for each unique value of the column. This leads to create thousands of partitions .

This can be avoided in Bucketing ,can limit the number of buckets to create.

A Bucket is a file whereas a partition is directory.

Buckets

- To create a bucketed table having 3 buckets.

```
CREATE TABLE IF NOT EXISTS STUDENT_BUCKET (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 the content of first bucket.

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

Views

View support is available only in version starting from 0.6.

To create a view table named “STUDENT_VIEW”

```
CREATE VIEW STUDENT_VIEW AS SELECT rollno, name FROM EXT_STUDENT;
```

Querying the view

```
SELECT * FROM STUDENT_VIEW LIMIT 4;
```

To drop the view

```
DROP VIEW STUDENT_VIEW;
```

Sub Querying

Write a sub query to count occurrence of similar words in the file

```
CREATE TABLE docs(line String);

LOAD DATA LOCAL INPATH ‘/root/hivedemos/lines.txt’ OVERWRITE INTO TABLE  docs;

CREATE TABLE word_count AS SELECT word, count(1) AS count FROM (SELCET    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 SQL joins

To create JOIN between Student and Department tables where we use RollNo from both the tables as the join key.

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

2. LOAD DATA LOCAL INPATH ‘/root/hivedemos/student.tsv’ OVERWRITWE INTO TABLE STUDENT;

3. 1.CREATE TABLE IF NOT EXISTS DEPARTMENT(rollno INT, deptno INT ,name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘\t’;

4. . LOAD DATA LOCAL INPATH ‘/root/hivedemos/department.tsv’ OVERWRITWE INTO TABLE DEPARTMENT;

5. SELECT a.rollno,a.name,a.gpa,b.deptno FROM STUDENT a JOIN DEPARTMENT b ON a.rollno=b.rollno

Aggregations

Hive supports aggregation functions like avg, count, etc.

```
. SELECT count(*) FROM STUDENT;
```

Group by and Having

To write group by and having function.

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

SerDer

- SerDer stands for Serializer/Deserializer.
- Contains the logic to convert unstructured data into records.
- Implemented using Java.
- Serializers are used at the time of writing.
- Deserializers are used at query time (SELECT Statement).

SERDE

To manipulate the XML data

```
        <employee>
                <empid>1001</empid>
                <name>John </name>
                <designation> TL</designation>
        </employee>
<employee>
                <empid>1002</empid>
                <name>Jack </name>
                <designation> PM</designation>
        </employee>
```

SERDE

```
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;
```

RCFile Implementation

```
 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;
```

USER DEFINED FUNCTIONS (UDF)

Write a hive function to convert the values of a field to upper case

```
Package com.example.hive.udf;
import org.apache.hadoop.hive.ql.exec.Descripition;
import org.apache.hadoop.hive.ql.exec.UDF;
@Description(
name=“simple UDF example”)

public final class MyLowerCase extends UDF{
public String evalutae(final String word){
return word.toLowerCase();
}
}
```

UDF

Convert this java program into Jar:

```
ADD JAR /root/hivedemos/UpperCase.jar

CREATE TEMPORARY FUNCTION touppercase AS ‘com.example.hive.udf.MyUpperCase’;

SELECT TOUPPERCASE(name) FROM STUDENT;
```

Thank you