UNIT-4
HIVE

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

Q) What is Hive? Explain features of Hive.

Hive is data warehousing tool and is used to query structured data that was built on top of Hadoop for providing data summarization, query, and analysis.

Hive Provides HQL (Hive Query Language) which is similar to SQL. Hive compiles SQL queries into MapReduce jobs and then runs the job in the Hadoop cluster.

Features of Hive:

- It is similar to SQL
- HQL is easy to code
- Hive supports rich datatypes such as structs, lists and maps
- Hive supports SQL filters, group-by and order-by clauses
- Custom types and custom functions can be defined.

Q) Explain various Hive Data Units

Databases: The name space 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.

Q) Explain Hive Architecture with a neat diagram.

Externel Interfaces- CLI, WebUI, JDBC, ODBC programming interfaces

Hive CLI: The most commonly used interface to interact with Hadoop.

Hive Web Interface: It is simple graphic interface to interact with Hive and to execute query.

Thrift Server – Cross Language service framework . This is an optional Sever. This can be used to submit Hive jobs from a remote client.

JDBC/ODBC: Jobs can be submitted from a JDBC client. One can write java code to connect to Hive and submit jobs on it.

Metastore- Meta data about the Hive tables, partitions. A metastore consists of Meta store service and Database.

There are three kinds of Metastore:

1. Embedded Meta store:This metastore is mainly used for unit tests. Here, only one process is allowed to connect to the metastore at a time. This is the default metastore for Hive. It is apache derby database.
2. Local Metastore: Metadata can be stored in any RDBMS component like MySQL. Local network allows multiple connections at a time. In this mode, Hive metastore service runs in the main Hive server process, but the metastore database runs in a separate process and can be on a separate host.

3. Remote Metastore: Hive driver and the metastore interface runs on different JVM's.

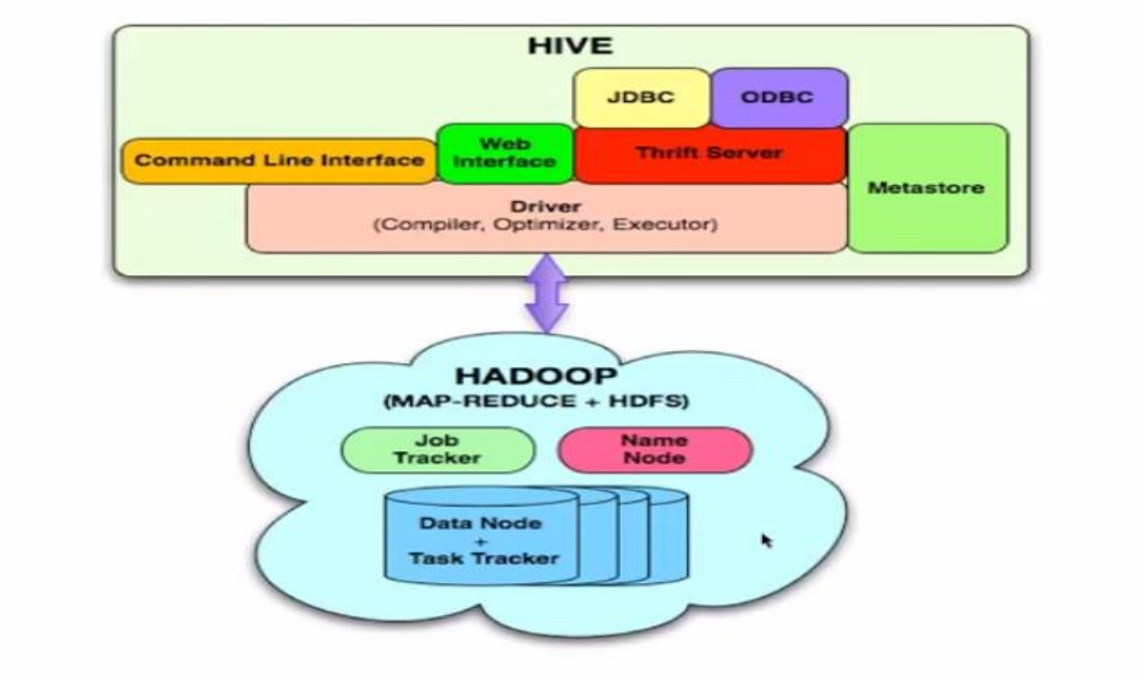

Fig. Hive Architecture

Driver- Brain of Hive! Hive queries are sent to the driver for Compilation, Optimization and Execution.

Q)Explain different data types in Hive.

Hive Data types are used for specifying the column/field type in Hive tables.
Mainly Hive Data Types are classified into 5 major categories, let's discuss them one by one:

a. Primitive Data Types in Hive
Primitive Data Types also divide into 3 types which are as follows:

- Numeric Data Type
- Date/Time Data Type
- String Data Type

TINYINT '1 byte signed integer', -128 to 127
SMALLINT '2 byte signed integer', -32, 768 to 32, 767
INT '4 byte signed integer', –2,147,483,648 to 2,147,483,647
BIGINT '8 byte signed integer', –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
FLOAT 'Single precision floating point',
DOUBLE 'Double precision floating point',
DECIMAL 'Precise decimal type based on Java BigDecimal Object',
TIMESTAMP 'YYYY-MM-DD HH:MM:SS.fffffffff" (9 decimal place precision)',
BOOLEAN 'TRUE or FALSE boolean data type',
STRING 'Character String data type',
BINARY 'Data Type for Storing arbitrary

b. Complex Data Types in Hive

In this category of Hive data types following data types are come-

- Array
- MAP
- STRUCT
- UNION

ARRAY<TINYINT> 'A collection of fields all of the same data type indexed BY an integer'

MAP<STRING,INT> 'A Collection of Key,Value Pairs where the Key is a Primitive Type and the Value can be anything. The chosen data types for the keys and values must remain the same per map'

STRUCT<first : SMALLINT, second : FLOAT, third : STRING>

'A nested complex data structure'

UNIONTYPE<INT,FLOAT,STRING>

'A Complex Data Type that can hold One of its Possible Data Types at Once'

eg.

t1.txt

1001,cse^1,1|2|3,A|50000,3,true
1002,cse^2,1|2,B|40000,good,true

Creating a table t1:

```
create table t1(id int,class map<string,int>,sections array<int>,hostel
struct<grade:string,fee:double>,rating uniontype<int,string>,exist boolean)
row format delimited
fields terminated by ','
collection items terminated by '|'
map keys terminated by '^'
lines terminated by '\n'
stored as textfile;
```

Q) Explain different HIVE file formats

The file formats in Hive specify how records are encoded in a file.

The file formats are :

1. **Text File:** The default file format is text file. In this format, each record is a line in the file.
2. **Sequential file:** Sequential files are flat files that store binary key-value pairs. It includes compression support which reduces the CPU, I/O requirement.
3. **RC File (Record Columnar File):** RCFile stores the data in column oriented manner which ensures that Aggregation operation is not an expensive operation.

   RC File Instead of only partitioning the table harizontally like the row oriented DBMS, RCFile partitions this table first horizontally and then vertically to serialize the data.

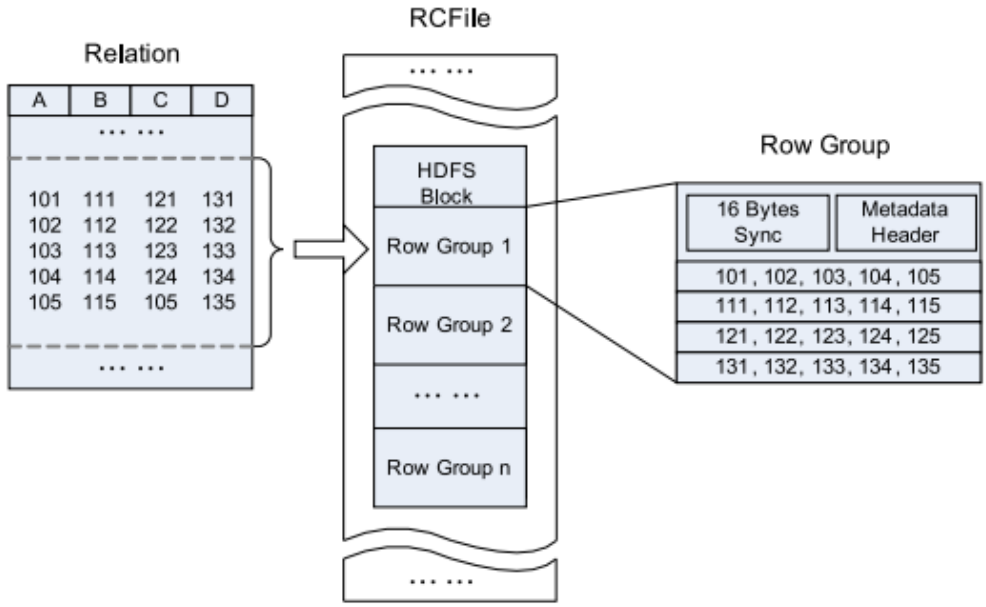

Fig. Hive RCFile Format

Q) What is Hive Query Language(HQL)? Explain various DDL and DML statements in Hive

Hive query language provides basic SQL like operations.
Basic tasks of HQL are:

1. Create and Manage tables and partitions
2. Support various relational, arithmentic and logic operations
3. Evaluate functions
4. Down load the contents of a table to a local directory or result of queries to HDFS directory.

HIVE DDL Statements:

These statements are used to build and modify the tables and other objects in the database.

1. Create/Drop/Alter Database
2. Create/Drop/truncate Table
3. Alter Table/partition/column
4. Create/Drop/Alter view
5. Create/Drop/Alter index
6. Show
7. Describe

HIVE DML Statements:

These statements are used to retrive, store, modify, delete and update data in database. The DML commands are as follows:

1. Loading files into table.
2. Inserting data into Hive tables from queries.

1. **Creating and Managing Databases and Tables**

hive> create database chp;
OK
Time taken: 0.116 seconds

2. TO CREATE DATABASE WITH COMMENTS AND DATABASE PROPERTIES

hive> create database if not exists chp comment 'employee details' with dbproperited('creator'='praneeth');

3. To display/list all the existing databases

hive> show databases;
OK
chp
default

4. To use created database chp

hive> use chp;
OK
Time taken: 0.018 seconds

5. To display DB NAME, COMMENT AND DB DIRECTORY

hive> describe database chp;
NOTE: SHOWS ONLY

6. To SHOWS DB PROPERTIES ALSO

hive> describe database extended chp;

7. To drop a database chp

hive> drop database chp;

8. Create tables emp and dept and load data from text files on hdfs.

Creating directory in hdfs and placing text files in hdfs:
hdfs dfs -mkdir /chp/data
hdfs dfs -put /home/vignan/Desktop/hive_data/*.txt /chp/data

contents in text file:

emp.txt
1,chp,10000.0
2,pnr,20000.0
3,kry,30000.0

fac.txt
2@cse
3@mca
4@cse

hive> create table emp(id int,name string,sal double) row format delimited fields terminated by ',';
OK
Time taken: 8.331 seconds

hive> show tables;
OK
emp

hive> create table dept(eid int,dept string) row format delimited fields terminated by '@';
OK

9. Loading data into the tables

hive> load data inpath '/chp/data/faculty.txt' into table emp;
hive> load data inpath '/chp/data/dept.txt' into table dept;

10. Retreiving data from the tables.

hive> select * from emp;

OK
1 chp 10000.0
2 pnr 20000.0
3 kry 30000.0
Time taken: 0.379 seconds, Fetched: 3 row(s)

hive> select * from dept;

OK
2 cse
3 mca
4 cse
Time taken: 0.133 seconds, Fetched: 4 row(s)

Queries:

1. Display different department in dept

```
hive> select distinct(dept) from dept;
cse
mca
```

2. Find the employees who earns 10000

```
hive> select name from emp where sal=10000;
chp
```

3. Find the employees who earns greater than 20000

```
hive> select name from emp where sal>=20000;
pnr
kry
```

4. Find the employee id whose name is either chp or kry

```
hive> select id from emp where name='chp' or name='kry';
1
3
```

5. Find the employee name whose dept is either cse or mca.

```
select emp.name from emp join dept on(emp.id=dept.eid) where
dept.dept='cse' or dept='mca';
pnr
kry
```

(or)

```
select emp.name from emp left outer join dept on(emp.id=dept.eid)
where dept.dept='cse' or dept.dept='mca';
pnr
kry
```

6. Find first 2 records in dept

```
hive> select * from dept limit 2;
OK
2 cse
3 mca
```

7. Find the no.of employees in each department.

```
hive> select dept,count(*) from dept group by dept;
cse 2
mca 1
```

8. Find the no.of employees in dept cse.

```
hive> select dept,count(*) from dept group by dept having dept='cse';
```

cse 2

9. **Find employees name and salary whose salary is greater than 20000.**

```
hive> select name,sal from emp where sal>=20000 limit 3;
pnr 20000.0
kry 30000.0
```

10. **Find the name of the employee who is earning minimum salary.**

```
hive> select name,sal from emp order by sal limit 1;
chp 10000.0
```

11. **Find the name of the employee who is earning maximum salary.**

```
hive> select name,sal from emp order by sal desc limit 1;
kry 30000.0
```

12. **Display all employee having sal greater than 10000**

```
hive>  select eid,name,sal from emp group by eid,name,sal having sal
> 10000;
```

Q) Briefly explain joins in Hive.

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database. It is more or less similar to SQL JOIN.

Inner join: The HiveQL INNER JOIN returns all the rows which are common in both the tables.

hive> select * from emp join dept on (emp.id=dept.eid);

```
2 pnr 20000.0 2 cse
3 kry 30000.0 3 mca
```

Left outer join: A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

hive> select * from emp left outer join dept on (emp.id=dept.eid);

```
1 chp 10000.0 NULL NULL
2 pnr 20000.0 2 cse
3 kry 30000.0 3 mca
```

Right Outer Join: The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

hive> select * from emp right outer join dept on (emp.id=dept.eid);

```
2 pnr 20000.0 2 cse
3 kry 30000.0 3 mca
NULL NULL NULL 4 cse
```

Full outer join: The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

hive> select * from emp full outer join dept on (emp.id=dept.eid);
1 chp 10000.0 NULL NULL
2 pnr 20000.0 2 cse
3 kry 30000.0 3 mca
NULL NULL NULL 4 cse

Q) Briefly explain about Views in Hive.
A view is purely a logical construct (an alias for a query) with no physical data behind it.
So altering a view only involves changes to metadata in the metastore database, not any data files in HDFS.
When a query becomes long or complicated, a view may be used to hide the complexity by dividing the query into smaller, more manageable pieces.

1. **Create a view from emp table with the fields id and name.**

hive> create view emp_view as select id,name from emp;
hive> select * from emp_view;
1 chp
2 pnr
3 kry

2. **Find no.of employees using above view.**

hive> select count(*) from emp_view;
3

3. **Drop view.**
   hive> drop view emp_view;

Q) Explain about various functions in Hive.
string functions:

1. **Display employee names in uppercase**

hive> select upper(name) from emp;
CHP
PNR
KRY

2. **Display employee names from 2nd character**

hive> select substr(name,2) from emp;
hp
nr
ry

3. **Concatenate emp id and name**

hive> select concat(id,name) from emp;
1chp
2pnr
3kry

Math Functions:

1. **Find the salaries of the employees by applying ceil function.**

```
hive> select ceil(sal) from emp;
10000
20000
30000
```

2. **Find the square root of the emp salaries.**

```
hive> select sqrt(sal) from emp;
100.0
141.4213562373095
173.20508075688772
```

3. **Find the length of the emp names.**

```
hive> select name,length(name) from emp;
chp 3
pnr 3
kry 3
```

Aggregate functions:

1. **Find no.of employees in the table emp.**

```
hive> select count(*) from emp;
3
```

2. **Find the salary of all the employees.**

```
hive> select sum(sal) from emp;
60000.0
```

3. **Find the average salary of the employees.**

```
hive> select avg(sal) from emp;
20000.0
```

4. **Find the minimum salary of all the employees.**

```
hive> select min(sal) from emp;
10000.0
```

5. **Find the maximum salary of all the employees.**

```
hive> select max(sal) from emp;
30000.0
```

Q) Differentiate Sort By and Order By in Hive

Hive sort by and order by commands are used to fetch data in sorted order. The main differences between sort by and order by commands are given below.

Sort by:

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

Order by:

Uses single reducer to guarantee total order in output.

1. **Display the employees according to their names using sort by.**

```
hive> select * from emp sort by name;
1 chp 10000.0
3 kry 30000.0
2 pnr 20000.0
```

2. Display the employees according to their names using order by.

```
hive> select * from emp order by name;
1 chp 10000.0
3 kry 30000.0
2 pnr 20000.0
```

Q) Write a query in hive to find word count in a text file. (or) Give an example for subquery in Hive

hello.txt(Input text file) :

```
hello welcome to guntur
hello welcome to vignan
welcome to cse
```

Creating a table for input:

```
hive> create table t2(st string);
```

Loading the given text data into the table:

```
hive> load data local inpath '/home/chp/Desktop/hello.txt' into table t2;
```

Displaying the content in the table:

```
hive> select * from t2;
hello welcome to guntur hello welcome to vignan welcome to cse
```

Wordcount Query:

hive> select word,count(*) as count from (SELECT explode(split(st,' ')) AS word FROM t2) temptable group by word;

Output:

```
cse 1
hello 2
 vignan 1
to 3
 guntur 1
welcome 3
```

explanation:

split returns an array of word for each line against given pattern.

Eg.

```
hive> select split(st,' ') from t2;
["hello","welcome","to","guntur"]
["hello","welcome","to","vignan"]
["welcome","to","cse"]
```

explode returns all the words i.e; in every line and store in temporary table.

Eg.

```
hive> select explode(split(st,' ')) as t3 from t2;
hello
welcome
to
guntur
hello
```

welcome
to
vignan
welcome
to
cse
here t3 is a temporary table.

Q) Explain about Index in Hive.

An Index acts as a reference to the records. Instead of searching all the records, we can refer to the index to search for a particular record.

In a Hive table, there are many numbers of rows and columns. If we want to perform queries only on some columns without indexing, it will take large amount of time because queries will be executed on all the columns present in the table.

Indexes are maintained in a separate table in Hive so that it won't affect the data inside the table, which contains the data.

Indexes are advised to build on the columns on which you frequently perform operations.

Building more number of indexes also degrade the performance of your query.

Types of Indexes in Hive
 Compact Indexing
 Bitmap Indexing

Differences between Compact and Bitmap Indexing
Compact indexing stores the pair of indexed column's value and its blockid.
Bitmap indexing stores the combination of indexed column value and list of rows as a bitmap.

Creating compact index:
Syntax:
hive> create index index_name on table table_name(columns,...) as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' with deferred rebuild;

Here, in the place of index_name we can give any name of our choice, which will be the table's INDEX NAME.
•In the ON TABLE line, we can give the table_name for which we are creating the index and the names of the columns in brackets for which the indexes are to be created. We should specify the columns which are available only in the table.
•The org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' line specifies that a built in CompactIndexHandler will act on the created index, which means we are creating a compact index for the table.
•The WITH DEFERRED REBUILD statement should be present in the created index because we need to alter the index in later stages using this statement.

Eg.

hive>create index emp_index on table emp(name,sal) as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' with deferred rebuild;

creating **bitmap index**:
Syntax:
create index index_name on table table_name(columns,...) as 'bitmap' with deferred rebuild;
eg.
hive> create index dept_index on table dept(eid) as 'bitmap' with deferred rebuild;

Q) Find different indices on table emp and dept
hive> show formatted index on emp;

idx_name tab_name col_names idx_tab_name idx_type
emp_index emp name, sal default__emp_emp_index__ compact

hive>show formatted index on dept;
idx_name tab_name col_names idx_tab_name idx_type
dept_index dept eid default__dept_dept_index__ bitmap

1) Update index emp_index
hive> alter index emp_index on emp rebuild;

2) Drop index emp_index
hive>drop index if exists emp_index on emp;

Q) Explain Partitioning in Hive.

In Hive, the query reads the entire dataset even though a where clause filter is specified on a particular column. This becomes a bottlenect in most of the MapReduce jobs as it involves huge degree of I/O.

So it is necessary to reduce I/O required by the MapReduce job to imporve the performance of the query. A very common method to reduce I/O is data partioning.

Partitions split the larger dataset into more meaningful chunks. Hive provides two kinds of partions.

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

Eg.
1) Create a partition table.
hive> create table std_partition(sid int) partitioned by (branch string) row format delimited fields terminated by ',' stored as textfile;

std1.txt
1001
1002

2) Load data into std_partition from st1.txt and partitioned column branch as cse.

```
hive> load data local inpath 'home/chp/Desktop/hive_data/std1.txt' into table std_partition partition(branch='cse');
hive> select * from std_partition;
OK
1001 cse
1002 cse
```

std2.txt
2001
2002

Loading data into std_partition from std2.txt and partitioned column branch as mca.

```
hive> load data local inpath '/home/chp/Desktop/hive_data/std2.txt' into table std_partition partition(branch='mca');

hive> select * from std_partition;
1001 cse
1002 cse
2001 mca
2002 mca
```

Dynamic partitioning: Dynamic partition have columns whose values are known only at Execution time.
By default the dynamic partitioning will be off. We can enable it by using the following commands in hive.

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> set hive.exec.max.dynamic.partitions.pernode=450;
```

1) Create a partition table.

```
hive> create table dept_partition(id int) partitioned by (branch string);
OK
Time taken: 0.105 seconds
```

2) Describe dept_partition.

```
hive> describe formatted dept_partition;
OK
# col_name data_type
id int None
# Partition Information
# col_name data_type
branch string None
```

3) Load data into dept_partition from dept table.

```
hive>insert into table dept_partition partition(branch) select * from dept;
hive> select * from dept_partition;

OK
```

2 cse
4 cse
3 mca

4) Drop partitioned table dept_partition.

```
hive> alter table dept_partition drop partition(branch='cse');
Dropping the partition branch=cse
OK
Time taken: 1.737 seconds
hive> select * from dept_partition;
OK
3 mca
```

Q) Explain how to create buckets in Hive.

Bucketing is similar to partition. However, there is a difference between partition and bucketing. In a partition, we need to create 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 we can limit the number of buckets to create. A bucket is a file whereas a partition is a directory.

Eg.

Set below property to enable bucketing.

```
set hive.enforce.bucketing=true;
```

Creating bucket on dept table:

create table dept_bucket(**eid int,dept string**) clustered by (**dept**) into 2 buckets;

loading data:

from dept insert overwrite table dept_bucket select eid,dept;

To display content from 1st bucket:

select distinct from dept_bucket tablesample(bucket 1 out 2 on dept);

Q) Explain about SERDE.

SerDe stands for Serializer/Deserializer.

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)

Deserializer interface takes a binary representation or string of a record, converts it into a jave object that hive can then manipulate. Serializer takes a java object that hive has been working with and translates it into something that hive can write to hdfs.

Input:

```
<employee>
        <empid>1001</empid>
        <name>chp</name>
        <designation>Analyst</designation>
</employee>
```

```
<employee>
        <empid>1002</empid>
        <name>praneeth</name>
        <designation>Team Lead</designation>
</employee>
```

create table xmlsample(xmldata string)
load data local inpath '/chp/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;

Output:
1001 chp Analyst
1002 praneeth Team Lead

Q) Explain about User Defined Function(UDF) in Hive with an example.

In Hive, we can use custom functions by defining the user defined functions Eg. to write a hive function to convert values of a field to lowercase.

```
package com.example.hive.udf;
import org.apavhe.hadoop.hive.ql.excec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
@Description(  name="SampleUDFExample")
public final class MyLowerCase extends UDF{
  public String evaluate(final String word){
        return word.toLowerCase();
        }
}
```

Note: We have to convert this java program into jar.

add jar /chp/hivedemo/UpperCase.jar
create temporary function tolowercase as
'com.example.hive.udf.MyLowerCase';
select touppercase(name) from emp;
output:
CHP
PNR
KRY

PIG

Introduction to Pig, The Anatomy of Pig , Pig on Hadoop , Pig Philosophy , Use Case for Pig: ETL Processing , Pig Latin Overview , Data Types in Pig , Running Pig , Execution Modes of Pig, HDFS Commands, Relational Operators, Piggy Bank , Word Count Example using Pig , Pig at Yahoo!, Pig versus Hive

Q) What is PIG? Explain key features of Pig.

Apache Pig is a platform for data analysis. It is an alternative to MapReduce Programming. Pig was developed as a research project at Yahoo.

Key Features:

1. It provides an engine for executing data flows(how the data should flow) Pig process data in parallel on the hadoop cluster.
2. It provides a language called "Pig Latin" to express data flows.
3. Pig Latin contains operators for many of the traditional data operations such as joins, filter, sort etc.
4. It allows users to develop their own functions (User Defined Functions) for reading, processing and writing data.
5. It supports HDFS commands, UNIX shell commands, Relational operators, mathematical functions and complex data structures.
6. **Ease of programming:** Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
7. **Optimization opportunities:** The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
8. **Handles all kinds of data:** Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Q) Explain the anatomy of PIG.

The main components of Pig are as follows:

1. Data flow language(Pig Latin)
2. Interactive shell where we can write Pig Latin Statements(Grunt)
   Eg.
   grunt> A = load 'student' (rollno, name, gpa);
   grunt>A = filter A by gpa4.0;
3. Pig interpreter and execution engine.
   - Processes and parses Pig Latin.
   - Checks data types.
   - Performs optimization.
   - Creates MapReduce Jobs.
   - Submits job to Hadoop
   - Monitors progress

Q) Explain PIG Philosophy

1. Pig eat's anything: Pig can process different kinds of data such as structured and unstructured data.
2. Pig live anything: Pig not only processes files in HDFS, it also processess files in other sources such as files in the local file system.
3. Pig is Domestic animal: Pig allows us to develop user defined functions and the same can be included in the script for complex operations
4. Pig Fly: Pig processes data quickly.

Q) Explain PIG ETL processing.

Pig widely used for ETL **(Extract, Transform and Load).** Pig can extract data from different sources such as ERP, accounting, flat files etc.. Pig then makes use of various operators to perform transformation the data and subsequently loads into the data warehouse.

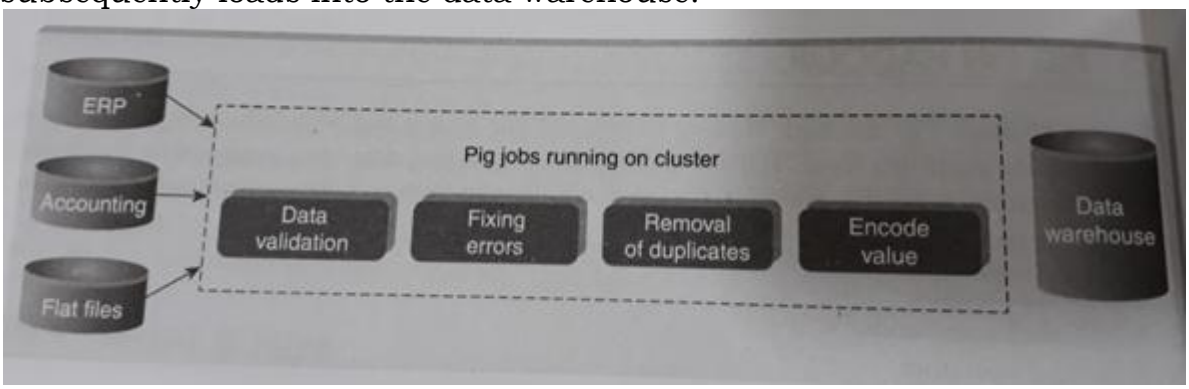

Fig. PIG ETL processing

Q) Explain PIG architecture with a neat diagram.

- The language used to analyze data in Hadoop using Pig is known as **Pig Latin**.
- It is a high-level data processing language which provides a rich set of data types and operators to perform various operations on the data.
- To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded).
- After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.
- Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.
- **Parser :** Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks.
  The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.
  In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.
- **Optimizer:** The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.
- **Compiler :**The compiler compiles the optimized logical plan into a series of MapReduce jobs.
- **Execution engine:** Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

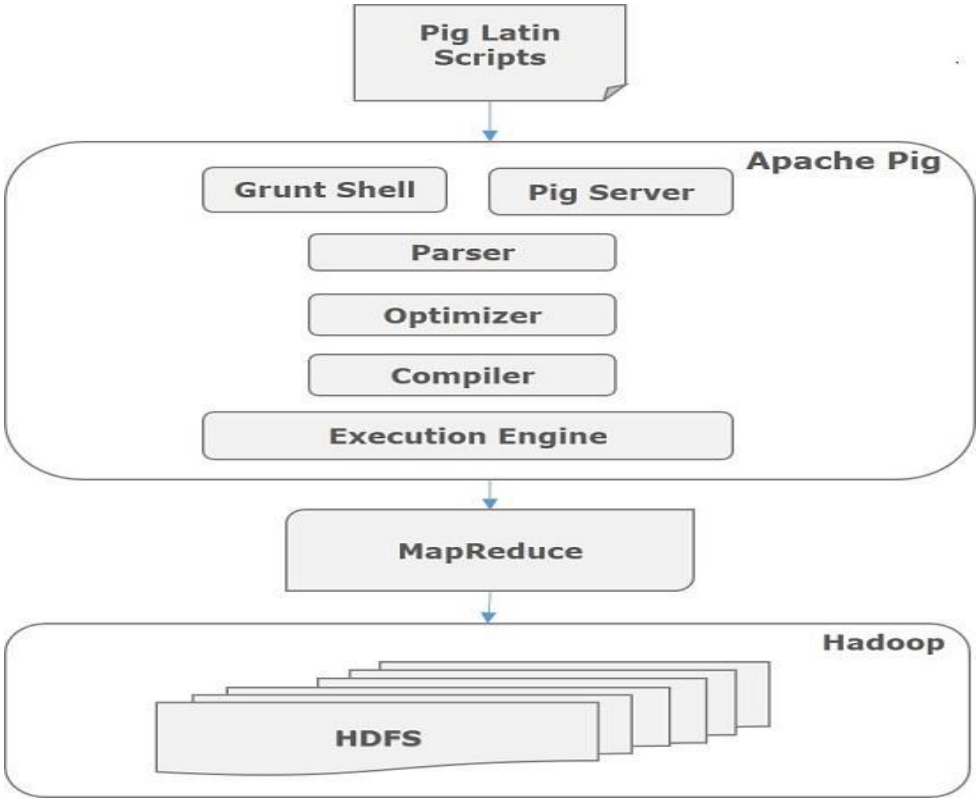

Fig. PIG Architecture

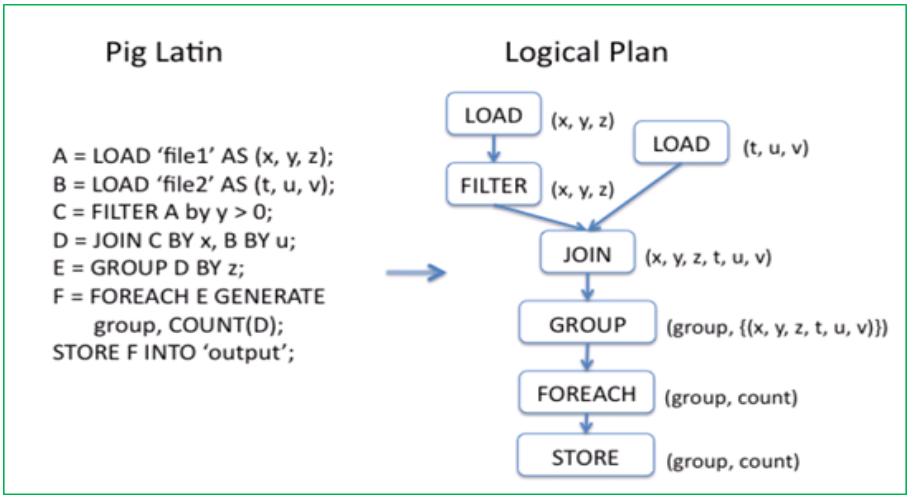

Fig. PIG Latin Script Execution

Q) Briefly discuss Pig Latin Application Flow.

At its core, Pig Latin is a dataflow language, where you define a data stream and a series of transformations that are applied to the data as it flows through your application. This is in contrast to a control flow language (like C or Java), where you write a series of instructions. In control flow languages, we use constructs like loops and conditional logic (like an if statement). We won't find loops and if statements in Pig Latin.

The syntax of a pig latin script is as follows:

1. **Load:** first load (LOAD) the data you want to manipulate. As in a typical MapReduce job, that data is stored in HDFS. For a Pig program to access the data, you first tell Pig what file or files to use. For that task, use the LOAD command.

The LOAD operator operates on the principle of lazy evaluation, also referred to as call-by-need.

The optional USING statement defines how to map the data structure within the file to the Pig data model — in this case, the PigStorage () data structure, which parses delimited text files.

The optional AS clause defines a schema for the data that is being mapped.If you don't use an AS clause, you're basically telling the default LOAD Func to expect a plain text file that is tab delimited.

2. **Transform:** run the data through a set of transformations that, which we have to concern ourself that are translated into a set of Map and Reduce tasks.

3. **Dump:** Finally, you dump (DUMP) the results to the screen or Store (STORE) the results in a file somewhere.

eg.

faculty.txt

```
1,chp,10000
2,pnr,20000
3,kry,10000
```

fac.pig

```
fac = load '/home/chp/Desktop/faculty.txt' using PigStorage(',') as
(id:int,name:chararray,sal:double,designation:chararray);
fac1 = filter fac by name=='chp';
dump fac1;
```

executing:

```
pig -x local
grunt> run fac.pig   or
grunt> exec fac.pig
```

Output:

```
(1,chp,10000.0,)
```

Q) Explain various Datatypes in Pig Latin.

Pig has a very limited set of data types. Pig data types are classified into two types. They are:

- Primitive
- Complex

Primitive Data Types: The primitive datatypes are also called as simple datatypes. The simple data types that pig supports are:

- int : It is signed 32 bit integer. This is similar to the Integer in java.
- long : It is a 64 bit signed integer. This is similar to the Long in java.
- float: It is a 32 bit floating point. This data type is similar to the Float in java.
- double: It is a 63 bit floating pint. This data type is similar to the Double in java.
- chararray: It is character array in unicode UTF-8 format. This corresponds to java's String object.

•bytearray: Used to represent bytes. It is the default data type. If you don't specify a data type for a filed, then bytearray datatype is assigned for the field.
•boolean: to represent true/false values.
Complex Types: Pig supports three complex data types. They are listed below:

✓ Atom: An atom is any single value, such as a string or a number Eg. int, long, float, double, chararray, and bytearray.
✓ Tuple: A tuple is a record that consists of a sequence of fields. Each field
can be of any type. Think of a tuple as a row in a table.
✓ Bag: A bag is a collection of non-unique tuples. The schema of the bag is
flexible — each tuple in the collection can contain an arbitrary number of fields, and each field can be of any type.
✓ Map: A map is a collection of key value pairs. Any type can be stored in the value, and the key needs to be unique. The key of a map must be a chararray and the value can be of any type.

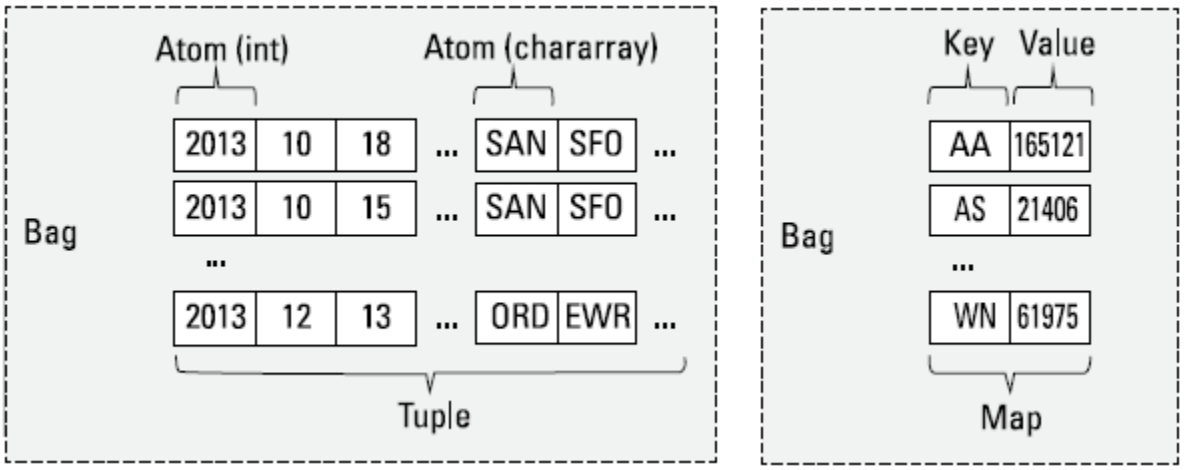

Fig. Pig Data Types

Q) Briefly discuss various modes in pig with an example.

Before you can run your first Pig script, you need to have a handle on how Pig programs can be packaged with the Pig server.
Pig has two modes for running scripts, as shown in Figure 3:
✓Local mode:
Local mode doesn't require Hadoop. When you run in Local mode, the Pig program runs in the context of a local Java Virtual Machine, and data access is via the local file system of a single machine. Local mode is actually a local simulation of MapReduce in Hadoop's LocalJobRunner class.

This can be useful for developing and testing Pig logic. If you're using a small set of data to develop or test your code, then local mode could be faster than going through the MapReduce infrastructure.
✓**MapReduce** mode (also known as Hadoop mode): Pig is executed on the Hadoop cluster. In this case, the Pig script gets converted into a series of MapReduce jobs that are then run on the Hadoop cluster.

Apache Pig scripts can be executed in three ways (**Execution Mechanisms**):

- **Interactive Mode** (Grunt shell) – You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).
- **Batch Mode** (Script) – You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with **.pig** extension.
- **Embedded Mode** (UDF) – Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java, and using them in our script.

Eg.

Executing pig in local mode:

To run a pig latin script in localmode write a script and save it on local file system. Let us assume script and data required is on Desktop.
Change present working directory to Desktop and enter into pig as below and execute.

faculty.txt
1,chp,10000
2,pnr,20000
3,kry,10000

1) Write a pig latin script to find details of the employee whose name is chp and run in local mode.

filter1.pig
```
fac = load '/home/chp/Desktop/faculty.txt' using PigStorage(',') as
(id:int,name:chararray,sal:double,designation:chararray);
fac1 = filter fac by name=='chp';
dump fac1;
```

executing:
```
pig -x local
grunt> run fac.pig   or
grunt> exec fac.pig
```

Output:
```
(1,chp,10000.0,)
```

Executing pig in mapreduce mode:

To run a pig latin script in hadoop mapreduce mode write a script and save it on hadoop file system. Similarly place necessary text files into the hdfs.

1) Write a pig latin script to group records by deptartment and run in mapreduce mode.

dept.txt
2@cse
3@mca
4@cse

group1.pig
```
dept1 = load '/user/chp/data/dept.txt' using PigStorage('@') as
(id:int,dept:chararray);
dept2 = group dept1 by dept;
dump dept2;
```

executing:
```
pig -x mapreduce
grunt> run dept.pig   or
grunt> exec dept.pig
```

Output:
```
(cse,{(2,cse),(4,cse)})
(mca,{(3,mca)})
```

Q) Explain about Pig Latin Relational Operators.

In a Hadoop context, accessing data means allowing developers to load, store, and stream data, whereas transforming data means taking advantage of Pig's ability to group, join, combine, split, filter, and sort data. Table 1 gives an overview of the operators associated
with each operation.

Pig Latin Operators		
Operation	***Operator***	***Explanation***
Data Access	LOAD/STORE	Read and Write data to file system
	DUMP	Write output to standard output (stdout)
	STREAM	Send all records through external binary
	FOREACH	Apply expression to each record and output one or more records
	FILTER	Apply predicate and remove records that don't meet condition
	GROUP/ COGROUP	Aggregate records with the same key from one or more inputs
	JOIN	Join two or more records based on a condition
Transformations	CROSS	Cartesian product of two or more inputs
	ORDER	Sort records based on key
	DISTINCT	Remove duplicate records
	UNION	Merge two data sets
	SPLIT	Divide data into two or more bags based on predicate
	LIMIT	subset the number of records

Table 1: Operators in pig

Pig also provides a few operators that are helpful for debugging and troubleshooting, as shown in Table 2

Operators for Debugging and Troubleshooting		
Operation	***Operator***	***Description***
Debug	DESCRIBE	Return the schema of a relation.
	DUMP	Dump the contents of a relation to the screen.
	EXPLAIN	Display the MapReduce execution plans.

Table 2: Debugging operators in pig.

Eg.

1) Write a pig latin script to find the highest salaried employee.

sort1.pig

```
fac = load '/home/chp/Desktop/faculty.txt' using PigStorage(',') as
(id:int,name:chararray,sal:double,designation:chararray);
fac1 = order fac by sal desc;
fac2 = limit fac1 1;
dump fac2;
```

executing:
pig -x local
grunt> run sort1.pig

output:
(2,pnr,20000.0,)

2) Write a pig latin script to display id and name of the employee.

Project1.pig
```
fac = load '/home/chp/Desktop/faculty.txt' using PigStorage(',') as
(id:int,name:chararray,sal:double,designation:chararray);
fac4 = foreach fac generate (id,name);
dump fac4;
```

executing:
pig -x local
grunt> run project1.pig

Output:
((1,chp))
((2,pnr))
((3,kry))

Q) Write a script for performing various join operations in pig.

joins.pig

```
fac = load '/home/chp/Desktop/faculty.txt' using PigStorage(',') as
(id:int,name:chararray,sal:double,designation:chararray);
dept1 = load '/home/chp/Desktop/dept.txt' using PigStorage('@') as
(id:int,dept:chararray);
j1 = join fac by id,dept1 by id;
dump j1;

j2 = join fac by id left outer,dept1 by id;
dump j2;

j3 = join fac by id right outer,dept1 by id;
dump j3;

j4 = join fac by id full outer,dept1 by id;
dump j4;
```

output:
inner join:
(2,pnr,20000.0,,2,cse)
(3,kry,10000.0,,3,mca)

left outer join:
(1,chp,10000.0,,,)
(2,pnr,20000.0,,2,cse)
(3,kry,10000.0,,3,mca)

right outer join:
(2,pnr,20000.0,,2,cse)
(3,kry,10000.0,,3,mca)
(,,,,4,cse)

full outer join:
(1,chp,10000.0,,,)
(2,pnr,20000.0,,2,cse)
(3,kry,10000.0,,3,mca)
(,,,,4,cse)

Q) Write the pig commands in local mode for the following queries using the realations(tables) faculty (fac) and department (dept1).

Entering into local mode: pig -x local

```
grunt> fac = load '/home/chp/Desktop/faculty.txt' using PigStorage(',') as (id:int,name:chararray,sal:double,designation:chararray);

grunt> dept1 = load '/home/chp/Desktop/dept.txt' using PigStorage('@') as (id:int,dept:chararray);

grunt> dump fac;

(1,chp,10000.0,)
(2,pnr,20000.0,)
(3,kry,10000.0,)

grunt> dump dept1;

(2,cse)
(3,mca)
(4,cse)
```

1. **Perform cross operation on fac and dept1**.

```
grunt> fac3 = cross fac,dept1;
grunt> dump fac3;
```

output:

```
(1,chp,10000.0,,2,cse)
(1,chp,10000.0,,3,mca)
(1,chp,10000.0,,4,cse)
(2,pnr,20000.0,,2,cse)
(2,pnr,20000.0,,3,mca)
(2,pnr,20000.0,,4,cse)
(3,kry,10000.0,,2,cse)
(3,kry,10000.0,,3,mca)
(3,kry,10000.0,,4,cse)
```

```
grunt> fac4 = cross dept1,fac;
grunt> dump fac4;
```

output:

```
(2,cse,1,chp,10000.0,)
(2,cse,2,pnr,20000.0,)
(2,cse,3,kry,10000.0,)
(3,mca,1,chp,10000.0,)
(3,mca,2,pnr,20000.0,)
(3,mca,3,kry,10000.0,)
(4,cse,1,chp,10000.0,)
(4,cse,2,pnr,20000.0,)
(4,cse,3,kry,10000.0,)
```

2. **Find the lowest salaried employee.**

```
grunt> fac5 = order fac by sal;
grunt> fac6 = limit fac5 1;
grunt> dump fac6;
```

output:

```
(1,chp,10000.0,)
```

3. **Find distinct departments in dept.**

```
grunt> dept2 = foreach dept1 generate dept;
grunt> dept3 = distinct dept2;
grunt> dump dept3;
```

output:

```
(cse)
(mca)
```

4. **Perform union operation of fac and dept1 relations.**

```
grunt> fd = union fac,dept1;
grunt> dump fd;
```

Output:

```
(1,chp,10000.0,)
(2,cse)
(2,pnr,20000.0,)
(3,mca)
(3,kry,10000.0,)
(4,cse)
```

5. **Split dept1 into various relations based on departments.**

```
grunt> split dept1 into dept2 if dept=='cse', dept3 if (dept=='mca' or dept=='it');

grunt> dump dept2;
```

Output:

```
(2,cse)
(4,cse)
```

grunt> dump dept3;

Output:
(3,mca)

6. Explain cogroup with an example.

we can use COGROUP when you need to group two tables by a column and then join on the grouped column.
Eg.

fac.txt	**dept.txt**
1,chp,10000	2,it
2,gokul,20000	3,cse
3,balaji,30000	4,it

dept = load '/home/chp/Desktop/dept.txt' PigStorage(',') as (id:int,chararry:dname);

fac = load '/home/chp/Desktop/fac.txt' PigStorage(',') as (id:int,chararray:name,double:sal);

grunt> cg = cogroup fac by id, dept by id;

dump cg;

Output:

(1,{(1,chp,10000.0)},{})

(2,{(2,gokul,20000.0)},{(2,it)})

(3,{(3,balaji,30000.0)},{(3,cse)})

(4,{},{(4,it)})

7. WordCount

hello.txt (Input text file)

hello welcome to Guntur
hello welcome to vignan
welcome to cse

```
grunt> text = load '/home/chp/Desktop/hello.txt' as (st:chararray);
grunt> words = foreach text generate FLATTEN(TOKENIZE(st,' ')) as word;
grunt> grouped = group words by word;
grunt> wordcount = foreach grouped generate group,COUNT(words);
grunt> dump wordcount;
```

(to,3)
(cse,1)
(hello,2)
(vignan,1)
(welcome,3)
(guntur,1)

Explanation:
Convert the Sentence into words: The data we have is in sentences. So we have to convert that data into words using TOKENIZE Function.

Output will be like this:
{(hello),(welcome),(to),(guntur)}
{(hello),(welcome),(to),(vignan)}
{(welcome),(to),(cse)}

Using FLATTEN function the bag is converted into tuple, means the array of strings converted into multiple rows.

Then the ouput is like below:

(hello)
(welcome)
(to)
(guntur)
(hello)
(welcome)
(to)
(vignan)
(welcome)
(to)
(cse)

We have to count each word occurrence, for that we have to group all the words as below:
Grouped = GROUP words BY word;
and then Generate word count as below:
wordcount = FOREACH Grouped GENERATE group, COUNT(words);

8. Describe fac
grunt> describe fac;
fac: {id: int,name: chararray,sal: double,designation: chararray}

Q) What is Pigg Bank.

Pig user can use Piggy Bank functions in Pig Latin script and they can also share their functions in Piggy Bank.
Eg. I/P: Student (rollno:int, name: chararray, gpa:float)

register '/root/pigdemos/piggybank-0.12.0.jar'
A = load 'pigdemos/student.txt' as (rollno:int, name:chararray, gpa:float);
upper = foreach A generate
org.apache.pig.piggybank.evaluation.string.UPPER(name);
dump upper;

Output:
JOHN
JACK

Q) Explain how to write a User Defined Function in PIG.

PIG allows us to create own function for complex analysis.
Java code to convert name into uppercase:

```
Package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;

public class UPPER extends EvalFunc<String>
{
  public String exec(Tuple input) throws IOException{
        if(input == null || input.size() ==0)
                return null;
        try{
                String str = (String)input.get(0);
                return str.toUpperCase();
        }catch(Exception e){
        throw WrappedIOException.wrap("Exception caught for input row processing",e);
            }
      }
}
```

Note: Convert above java class into jar to include this function into our code.
I/P: Student (rollno:int, name: chararray, gpa:float)

```
register '/root/pigdemos/piggybank-0.12.0.jar'
A = load 'pigdemos/student.txt' as (rollno:int, name:chararray, gpa:float);
B = foreach A generate myudfs.UPPER(name);
Dump B;
```

Output:
JOHN
JACK

Q) Differentiate PIG and HIVE

Features	**Pig**	**Hive**
Used by	Programmers and Researchers	Analyst
User for	Programming	Reporting
Language	Prodedural data flow language	SQL like
Suitable for	Semi-structured and Unstructured	Structured
Schema/Types	Explicit	Implicit
UDF Support	YES	YES
Join/Order/Sort	YES	YES
DFS Direct Access	Implicit	Explicit
Web Interface	YES	NO
Partitions	YES	YES
Shell	YES	YES

Q) Differentiate PIG and MapReduce.

Apache Pig	**MapReduce**
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.

Important Questions

1. a. Explain HIVE architecture in detail.

   b. Explain PIG architecture in detail.

2. Write HQL statements to create join between student(RollNo,SName,gpa) and department(RollNo,dname) tables where we use RollNo from both the tables as the join key.

3. a. Write HQL sub-query to count occurrence of similar words in the file.

   b. Write a word count program in Pig to count the occurrence of similar words in a file.

4. A. Discuss various data types in Pig.

   B. Discuss various data types in Hive.

5. a. Explain in detail how Hive is different from Pig.

   b. Differentiate hive and mapreduce.

6. Perform the following operations using Hive Query language

   Create a database named “STUDENTS” with comments and database properties,

   Display a list of databases

   Describe a database

   To make the databases current working database

   To delete or remove a database.

7. (a) Explain about partitioning in Hive.

   (b) Explain about bucketing in Hive.

9. Explain how to create a user defined functions in Hive and Pig with an example for each.

10. Explain the following with an example

a. Piggy Bank

b. Modes of Pig

c. Execution mechanisms of Pig