

BIG DATA ANALYTICS USING HP VERTICA-2 NHOP04

Topics:

- Vertica cluster management
- Hadoop Hands on sessions
- SQOOP Hands on sessions

Vertica Cluster Management

- Adding nodes to an existing cluster
- Removing nodes from a cluster
- Replacing nodes
- Rebalancing data across nodes

Adding nodes to an existing cluster:

- There are many reasons for adding one or more nodes to an installation of Vertica:
 1. **Increase system performance:**
Add additional nodes due to a high query load or increase disk space without adding storage locations to existing nodes.
 2. **Make the database K-safe > 1**
 3. **Swap a node for maintenance.** Use a spare machine to temporarily take over the activities of an existing node that needs maintenance. The node that requires maintenance is known ahead of time so that when it is temporarily removed from service, the cluster is not vulnerable to additional node failures.
 4. **Replace a node.** Permanently add a node to remove obsolete or malfunctioning hardware.

Adding nodes consists of the following general tasks:

1. Back up the database

HPE strongly recommends that you back up the database before you perform this significant operation because it entails creating new projections, refreshing them, and then deleting the old projections.

2. Configure the hosts you want to add to the cluster.

You will also need to edit the hosts configuration file on all of the existing nodes in the cluster to ensure they can resolve the new host.

3. Add one or more hosts to the cluster

After you have backed up the database and configured the hosts you want to add to the cluster

4. Add the hosts you added to the cluster (in step 3) to the database.

- Once you have added one or more hosts to the cluster, you can add them as nodes to the database.

Note: When you add a "host" to the database, it becomes a "node." You can add nodes to your database using either the Administration Tools or the Management Console

Removing nodes from a cluster:

- Although less common than adding a node, permanently removing a node is useful if the host system is obsolete or over-provisioned.
- You cannot remove nodes if your cluster would not have the minimum number of nodes required to maintain your database's current K-safety.
- If you really wish to remove the node or nodes from the database, you first must reduce the K-safety level of your database.

- Removing one or more nodes consists of the following general steps:

1. Back up the database

HPE recommends that you back up the database before performing this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. Lower the K-safety of your database

if the cluster will not be large enough to support its current level of K-safety after you remove nodes.

3. Remove the nodes from the database.

4. Remove the hosts from the cluster

if they are not used by any other databases.

Replacing nodes:

- If you have a K-Safe database, you can replace nodes, as necessary, without bringing the system down.
- For example, you might want to replace an existing node if you:
 - Need to repair an existing host system that no longer functions and restore it to the cluster
 - Want to exchange an existing host system for another more powerful system

Note: Vertica does not support replacing a node on a K-safe=0 database.

- The process you use to replace a node depends on whether you are replacing the node with:
 - A host that uses the same name and IP address
 - A host that uses a different name and IP address

Rebalancing data across nodes:

- Vertica can rebalance your database when you add or remove nodes. Manually trigger a rebalance by using Administration Tools, SQL functions, or using Management Console.
- For segmented projections, Vertica creates new (renamed), segmented projections that are identical in structure to the existing projections, but which have their data distributed across all nodes.
- The rebalance process then refreshes all new projections, and drops all of the old segmented projections.
- All new buddy projections have the same base name so they can be identified as a group.
- For unsegmented projections, leaves existing projections unmodified, creates new projections on the new nodes, and refreshes them.
- After the data has been rebalanced, Vertica drops:
 - Duplicate buddy projections with the same offset
 - Duplicate replicated projections on the same node
- Before data rebalancing completes, Vertica operates with the existing K-safe value. After rebalancing completes, Vertica operates with the K-safe value specified during the rebalance operation.

List of Hadoop & Sqoop programs

1. Checking hadoop configuration files.
2. Loading a file from local file system to hadoop file system.
3. Perform analysis on loaded files using hadoop mapreduce programs and verify the output using hadoop commands as well as browser.
 - (a) Count
 - (b) Grep
4. Verifying Sqoop status through cloudera manager
5. Hand-on Practice on various Sqoop basic commands
 - (a) List-database
 - (b) List-table
 - (c) Eval
6. Import of tables from Mysql database to hdfs
 - (a) Import of all tables
 - (b) Import of specific tables to default directory /target directory
 - (c) Import of subset of tables using 'where' clause
 - (d) Incremental import
7. Export files from hdfs to mysql database

Hadoop Hands on sessions

1. Checking hadoop configuration files.

(a) To verify that all softwares are in good health or not

- go to browser
- click on “Cloud Manager”
- enter Username: cloudera
Password: cloudera
- click on “login”

(b) To check java path

- open the terminal
- cd /usr/lib/jvm
- ls

(c) To check hadoop location

- cd /usr/lib/hadoop-0.20-mapreduce/
- ls

(d) To verify hadoop installation files

- cd conf
- ls
- gedit core-site.xml (similarly we can open other files)

2. Loading a file from local file system to hadoop file system.

- open the terminal
- to verify whether all daemons are running or not
`sudo jps`
- To create a folder or directory in hadoop
`hadoop fs -mkdir /user/cloudera/nh001`
- To verify whether or not the folder is created
`hadoop fs -ls /user/cloudera`
- To create a file in local file system
`gedit test`
enter some sample data in it and then save & close
- To verify whether or not the file is created
`ls`

- To put the local file into hadoop file system
`hadoop fs -put test /user/cloudera/nh001`
- To verify whether or not the local file is loaded into hadoop file system
`hadoop fs -ls /user/cloudera/nh001`
- To check the content of loaded file
`hadoop fs -cat /user/cloudera/nh001/test`

3. Perform analysis on loaded files using hadoop mapreduce programs and verify the output using hadoop commands as well as browser.

(a) Count

(b) Grep

- To see the list of jar files available in hadoop

 - `cd /usr/lib/hadoop-0.20-mapreduce/`

 - `ls`

- To see the content of jar file

 - `hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples-2.0.0-mr1-cdh4.4.0.jar`

- To run word count program on loaded file and creating output file path.

```
hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples-2.0.0-mr1-cdh4.4.0.jar wordcount /user/cloudera/nh001/test /user/cloudera/nh001/output1
```

- To verify output files

```
hadoop fs -ls /user/cloudera/nh001
```

```
hadoop fs -ls /user/cloudera/nh001/output1
```

- To see the content of output file

```
hadoop fs -cat /user/cloudera/nh001/output1/part-r-00000
```

- To see the output through browser

click on “HDFS Namenode” → Browse the file system → user → cloudera → nh001 → output1 → part-r-00000

(b) Grep

- To run Grep program on loaded file and creating output file path.

```
hadoop jar /usr/lib/hadoop-0.20-  
mapreduce/hadoop-examples-2.0.0-mr1-  
cdh4.4.0.jar grep /user/cloudera/nh001/test  
/user/cloudera/nh001/output2 key_word
```

- Remaining steps as before

SQOOP Hands on sessions

1. Verifying Sqoop status through cloudera manager

- open the terminal
- To start mysql services
`sudo service mysqld start`
- To connect to mysql
`mysql -u root -p`
Pass: root
- To create Database
`create database sampled;`
- To show the existing data bases
`show databases;`
- To drop database
`drop database databasename;`
- Exit

- To verify the database names
show databases;
- To choose the database you want to use
use sampled;db;
- To create tables
create table std(rno int);
create table emp(id int, name char);
- To insert records into the tables
insert into stud values (101), (102), (103), (104), (105);
insert into emp values (1,'a'), (2,'b'), (3,'c'), (4,'d'),
(5,'e');
- Exit
- Open the browser and select “Cloudera Manager”
- Check whether or not SQOOP is in good helath

2. Hand-on Practice on various Sqoop basic commands

(a) List-database (b) List-table (c) Eval

- (a) To list the databases

```
sqoop list-databases --connect "jdbc:mysql://localhost"  
--username root --password root
```

- (b) To list the tables

```
sqoop list-tables --connect  
"jdbc:mysql://localhost/sampledb"  
--username root --password root
```

- (c) To run sql queries from hadoop using eval

```
sqoop eval --connect "jdbc:mysql://localhost/sampledbs"
--username root --password root
--query "select * from emp"
```

```
sqoop eval --connect "jdbc:mysql://localhost/sampledbs"
--username root --password root
--query "select count(*) from stud"
```

```
sqoop eval --connect "jdbc:mysql://localhost/sampledbs"
--username root --password root
--query "select * from emp where id>2"
```

```
sqoop eval --connect "jdbc:mysql://localhost/sampledbs"
--username root --password root
--query "insert into emp values(5,'y')"
```

3. Import of tables from Mysql database to hdfs

- (a) Import of all tables
- (b) Import of specific tables to default directory /target directory
- (c) Import of subset of tables using 'where' clause
- (d) Incremental import

(a) Import of all tables:

```
sqoop import-all-tables --connect  
"jdbc:mysql://localhost/sampled"b"  
--username root --password root -m 1
```

To check whether or not tables are imported

```
hadoop fs -ls /user/cloudera
```

To check for a particular table

```
hadoop fs -ls /user/cloudera/stud
```

To see the records in a table

```
hadoop fs -cat /user/cloudera/stud/part-m-00000
```

Note: if name node is in safe mode import wont work

To remove the directory

```
hadoop fs -rm -R /user/cloudera/stud
```

```
hadoop fs -rm -R /user/cloudera/emp
```

(b) Import of specific tables to default directory /target directory

To import specific table to the default directory

```
sqoop import --connect "jdbc:mysql://localhost/sampled" --username root --password root --table emp -m 1
```

To check the imported table in the default directory

```
hadoop fs -ls /user/cloudera/
```

```
hadoop fs -ls /user/cloudera/emp
```

```
hadoop fs -cat /user/cloudera/emp/part-m-00000
```


To make a new directory

```
hadoop fs -mkdir /user/cloudera/hp2
```

To import specific table to the target directory

```
sqoop import --connect "jdbc:mysql://localhost/sampledatab"
--username root --password root --table emp
--target-dir /user/cloudera/hp2/sqoopplab1 -m 1
```

To check the imported table in the target directory

```
hadoop fs -ls /user/cloudera/hp2
```

```
hadoop fs -ls /user/cloudera/hp2/sqoopplab1
```

```
hadoop fs -cat /user/cloudera/hp2/sqoopplab1/part*
```

Importing single or multiple tables to specific directory

```
sqoop import-all-tables --connect  
  "jdbc:mysql://localhost/sampledatab"  
  --username root --password root  
  --warehouse-dir /user/cloudera/hp2 -m 1
```

```
sqoop import --connect "jdbc:mysql://localhost/sampledatab"  
  --username root --password root  
  --table emp  
  --warehouse-dir /user/cloudera/hp2 -m 1
```

(c) Import of subset of tables using 'where' clause

To import subset of data

```
sqoop import --connect "jdbc:mysql://localhost/sampledbs"
--username root --password root
--table emp --where "id>'2'"
--target-dir /user/cloudera/hp2/sqooplab2 -m 1
```

To check the imported table in the target directory

```
hadoop fs -ls /user/cloudera/hp2
```

```
hadoop fs -ls /user/cloudera/hp2/sqooplab2
```

```
hadoop fs -cat /user/cloudera/hp2/sqooplab2/part*
```

Import of subset of tables using 'where' clause and columns

```
sqoop import --connect  
"jdbc:mysql://localhost/sampledb"  
--username root --password root  
--table emp --columns "col1,col2"  
--where "id>'2'"  
--target-dir  
/user/cloudera/hp2/sqoopplab2 -m 1
```

(d) Incremental import

- To insert new records into the table

```
sqoop eval --connect "jdbc:mysql://localhost/sampledbs"  
--username root --password root  
--query "insert into stud values (106),(107)"
```

- To import new records into hadoop

```
sqoop import --connect "jdbc:mysql://localhost/sampledbs"  
--username root --password root --table stud  
--target-dir /user/cloudera/hp2/sqoopplab1  
--incremental append --check-column rno  
--last-value 105 --m 1
```

4. Export files from hdfs to mysql database

- To create a file in local file system

```
gedit test
```

```
1,a
```

```
2,b
```

```
3,c
```

```
save & exit
```

- To put the local file into hadoop file system

```
hadoop fs -put test /user/cloudera/hp2
```

- To verify whether or not the local file is loaded into hadoop file system

```
hadoop fs -ls /user/cloudera/hp2
```

- To check the content of loaded file

```
hadoop fs -cat /user/cloudera/hp2/test
```

- To create a table structure in mysql

```
sqoop eval --connect "jdbc:mysql://localhost/sampledatab"
--username root --password root
--query "create table test_table(a int,b char)"
```

- To export the file from hadoop to mysql

```
sqoop export --connect "jdbc:mysql://localhost/sampledatab"
--username root --password root --table test_table
--export-dir /user/cloudera/hp2/test
```

- To check the table data in mysql

```
sqoop eval --connect "jdbc:mysql://localhost/sampledatab"
--username root --password root
--query "select * from test_table"
```

Importing single or multiple tables to specific directory

```
sqoop import --connect  
"jdbc:mysql://localhost/sampledbs"  
--username root --password root  
--table emp  
--warehouse-dir /user/cloudera/hp2 -m 1
```



```
sqoop import-all-tables --connect  
  "jdbc:mysql://localhost/sampledb"  
  --username root --password root  
  --warehouse-dir /user/cloudera/hp2 -m 1
```

Note:

- -m 1 option is necessary when there is no primary key to the table
- -m n you can increase the mappers if you have primary key to the table
- --target-dir will not work on importing all the tables.(you are given choice to mention the directory name where as in warehouse-dir directories are created with same name as mysql table.

Module – 2 (PIG)

- Apache Pig Architecture
- Pig Latin Data Model
- Apache Pig Execution Modes
- Data types

Hands on sessions:

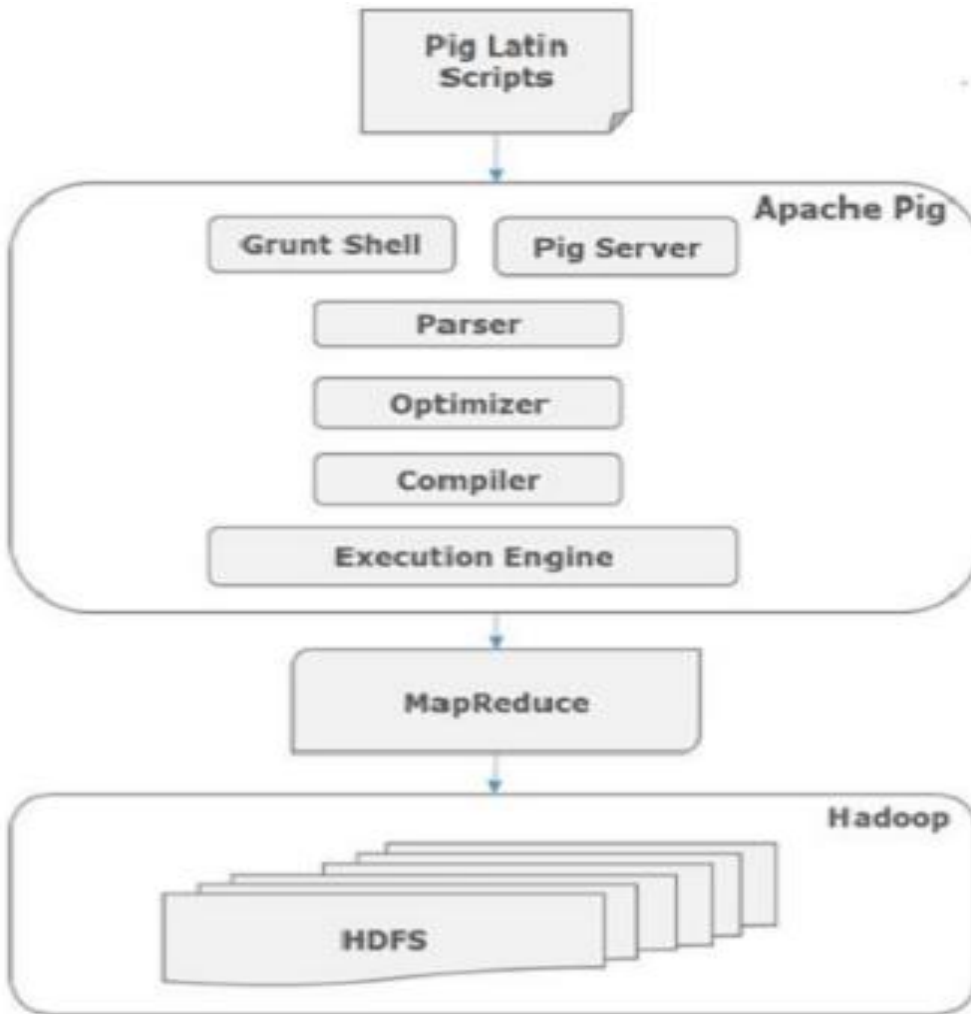
1. Loading data from local file system/hdfs to PIG and vice versa.
2. Diagnostic operator – Dump, Describe, Explain, illustrate

3. Filtering – filter operator; For Each Generate operator: projection, nested projection; Distinct Operator
4. Arithmetic operators, Comparison operator, Boolean Operators

Introduction:

- Apache Pig is designed to reduce the complexities of coding Map-Reduce applications.
- 10 lines of code in Pig equal 200 lines of code in Java.
- It is used to load the data, apply the required filters and dump the data in the required format
- To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language.
- All these scripts are internally converted to Map and Reduce tasks.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Apache Pig - Architecture



Apache Pig Components:

- As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

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.

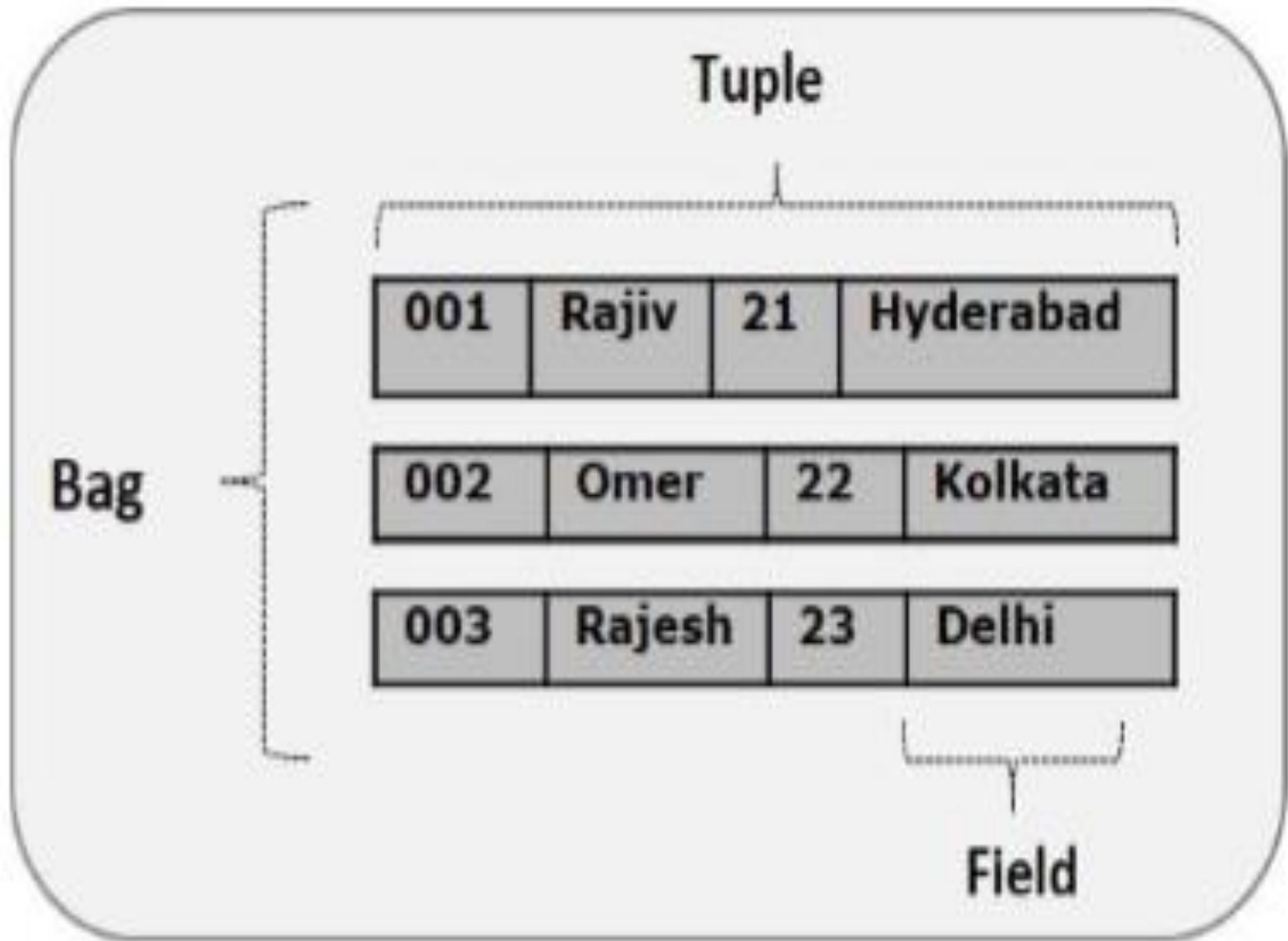
Compiler

- 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.

Pig Latin Data Model



Pig Latin Data Model

Atom: Any single value in Pig Latin, irrespective of their data type is known as an **Atom**.

- ✓ A piece of data or a simple atomic value is known as a **field**.
- ✓ **Example** – 'raja' or '30'

Tuple: A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type.

- ✓ A tuple is similar to a row in a table of RDBMS.
- ✓ **Example** – (Raja, 30)

Bag: A bag is an unordered set of tuples.

- ✓ In other words, a collection of tuples is known as a bag.
- ✓ Each tuple can have any number of fields (flexible schema).
- ✓ A bag is represented by '{}'. It is similar to a table in RDBMS.
- ✓ **Example** – {(Raja, 30), (Mohammad, 45)}

Map: A map (or data map) is a set of key-value pairs.

- ✓ The **key** needs to be of type chararray and should be unique.
- ✓ The **value** might be of any type. It is represented by '[]'
- ✓ **Example** – [name#Raja, age#30]

Apache Pig Execution Modes

Pig has two execution modes:

Local Mode

- In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

MapReduce Mode

- MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

Invoking the Grunt Shell

You can invoke the Grunt shell in a desired mode (local/MapReduce) using the **-x** option as shown below.

Local mode: **pig -x local**

Mapreduce mode: **pig -x mapreduce** or **pig**

- ✓ Either of these commands gives you the Grunt shell prompt as shown below.

grunt>

- ✓ You can exit the Grunt shell using '**ctrl + d**' or '**quit**'

Data Types

S.No.	Data Type	Description & Example
1	int	Represents a signed 32-bit integer. Example : 8
2	float	Represents a signed 32-bit floating point. Example : 5.5F
3	double	Represents a 64-bit floating point. Example : 10.5
4	chararray	Represents a character array (string) in Unicode UTF-8 format. Example : 'tutorials point'
5	Boolean	Represents a Boolean value. Example : true/ false.
6	Datetime	Represents a date-time. Example : 1970-01-01T00:00:00.000+00:00
7	BigDecimal	Represents a Java BigDecimal Example : 185.98376256272893883

Complex data types

S.No.	Data Type	Description & Example
1	Tuple	A tuple is an ordered set of fields. Example : (raja, 30)
2	Bag	A bag is a collection of tuples. Example : {(raju,30),(Mohhammad,45)}
3	Map	A Map is a set of key-value pairs. Example : ['name' #'Raju', 'age' #30]

Hands on sessions:

1. Loading data from local file system/hdfs to PIG and vice versa.
2. Diagnostic operator – Dump, Describe, Explain, illustrate
3. Filtering – filter operator; For Each Generate operator: projection, nested projection; Distinct Operator
4. Arithmetic operators, Comparison operator, Boolean Operators

Program – 1: Loading data from local file system/hdfs to PIG and vice versa.

Local \longleftrightarrow PIG

Open the terminal

To check the present working directory

```
[cloudera@localhost ~]$ pwd
```

```
/home/cloudera
```

To create a data file in local system

```
[cloudera@localhost ~]$ gedit datafile
```

```
1,a
```

```
2,b
```

```
3,c
```

```
4,d
```

Save and close

To verify the file

```
[cloudera@localhost ~]$ ls
```

```
[cloudera@localhost ~]$ cat datafile
```

```
1,a
```

```
2,b
```

```
3,c
```

```
4,d
```

To switch to local execution mode in PIG

```
[cloudera@localhost ~]$ pig -x local
```

```
grunt>
```

To clear the screen

```
grunt> clear
```

To load the data file into PIG

```
grunt> bag1 = load '/home/cloudera/datafile' using PigStorage(',') as  
              (id:int,name:chararray);
```

To see the content of the bag/relation

```
grunt> dump bag1;
```

To create alias

```
grunt> bag2 = bag1;
```

To see the content of the alias

```
grunt> dump bag2;
```

To store the result of bag into the local system

```
grunt> store bag1 into '/home/cloudera/output1' using PigStorage(',');
```

```
[main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

To store the result of bag onto the Desktop

```
grunt> store bag1 into '/home/cloudera/Desktop/output1' using PigStorage(',');
```

To verify the output file in local system

```
grunt> quit
```

```
[cloudera@localhost ~]$ ls
```

```
[cloudera@localhost ~]$ ls output1
```

```
part-m-00000 _SUCCESS
```

```
[cloudera@localhost ~]$ cat output1/part*
```

```
1,a
```

```
2,b
```

```
3,c
```

```
4,d
```

Also check the desktop for the output file.

HDFS \longleftrightarrow PIG

Loading datafile from HDFS to PIG

```
[cloudera@localhost ~]$ gedit datafile2
```

```
ramu,bangalore,20
```


raju,chennai,21
john,bangalore,21
sita,hyderabad,22
krish,delhi,20
anu,chennai,19 save and close

To put this file into hadoop

```
[cloudera@localhost ~]$ hadoop fs -put datafile2 /user/cloudera/
```

To verify the file in hadoop

```
[cloudera@localhost ~]$ hadoop fs -ls /user/cloudera/
```

```
[cloudera@localhost ~]$ hadoop fs -cat /user/cloudera/datafile2
```

ramu,bangalore,20

raju,chennai,21

john,bangalore,21

sita,hyderabad,22

krish,delhi,20

anu,chennai,19

To switch to mapreduce mode in PIG

```
[cloudera@localhost ~]$ pig -x mapreduce
```

Or

```
[cloudera@localhost ~]$ pig
```

```
grunt > clear
```

To load the file from hdfs to PIG

```
grunt> student = load '/user/cloudera/datafile2' using PigStorage(',') as  
                  (name:chararray, city:chararray,age:int);
```

To see the content

```
grunt> dump student;  
(ramu,bangalore,20)  
(raju,chennai,21)  
(john,bangalore,21)  
(sita,hyderabad,22)  
(krish,delhi,20)  
(anu,chennai,19)
```

To create alias and see the content

```
grunt> a = student;  
grunt> dump a;
```

To store the result into hdfs

```
grunt> store student into '/user/cloudera/output1' using PigStorage(',');
```

[main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

To check the output file in hadoop

```
grunt> fs -ls /user/cloudera/  
grunt> fs -ls /user/cloudera/output1  
grunt> fs -cat /user/cloudera/output1/part*  
ramu,bangalore,20  
raju,chennai,21  
john,bangalore,21  
sita,hyderabad,22  
krish,delhi,20
```

anu,chennai,19

We also check the result through the browser

HDFSNameNode-→Browse the file system→ user→ cloudera→Output1→part-m-00000

Program2: Diagnostic operator – Dump, Describe, Explain, illustrate

The **describe** operator is used to view the schema of a relation.

Syntax:

```
grunt> Describe Relation_name
```

```
grunt> describe student;
```

```
student: {name: chararray,city: chararray,age: int}
```

The **explain** operator is used to display the logical, physical, and MapReduce execution plans of a relation.

Syntax:

```
grunt> explain Relation_name;
```

```
grunt> explain student;
```

```
#New Logical Plan:
```

```
#Physical Plan:
```

```
# Map Reduce Plan
```

The **illustrate** operator gives you the step-by-step execution of a sequence of statements.

Syntax:

```
grunt> illustrate Relation_name;
```

```
grunt> illustrate student;  
grunt> illustrate student;  
grunt> illustrate student;
```

Program3: Filtering – filter operator; For Each Generate operator: projection, nested projection; Distinct Operator

The **FILTER** operator is used to select the required tuples from a relation based on a condition.

Syntax:

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

```
grunt> A = filter student by city == 'bangalore';  
grunt> dump A;
```

```
(ramu,bangalore,20)  
(john,bangalore,21)
```

The **FOREACH** operator is used to generate specified data.

Syntax:

```
grunt> Relation_name2 = FOREACH Relatin_name1 GENERATE (required data);
```

```
grunt> c = foreach student generate *;  
grunt> dump c;
```

(ramu,bangalore,20)
(raju,chennai,21)
(john,bangalore,21)
(sita,hyderabad,22)
(krish,delhi,20)
(anu,chennai,19)

```
grunt> c = foreach student generate age;  
grunt> dump c;  
(20)  
(21)  
(21)  
(22)  
(20)  
(19)
```

The **DISTINCT** operator is used to remove redundant (duplicate) tuples from a relation.

Syntax:

```
grunt> Relation_name2 = DISTINCT Relatin_name1;
```

```
grunt> a = distinct c;  
grunt> dump a;  
(19)  
(20)  
(21)  
(22)
```

The **GROUP** operator is used to group the data a relation. It collects the data having the same key.

Syntax:

```
grunt> Group_data = GROUP Relation_name BY field;
```

```
grunt> g = group student by city;
```

```
grunt> dump g;
```

```
(delhi,{{krish,delhi,20}})
```

```
(chennai,{{raju,chennai,21},{anu,chennai,19}})
```

```
(bangalore,{{ramu,bangalore,20},{john,bangalore,21}})
```

```
(hyderabad,{{sita,hyderabad,22}})
```

```
grunt> describe g;
```

```
g: {group: chararray,student: {{name: chararray,city: chararray,age: int}}}
```

Program 4: Arithmetic operators, Comparison operator, Boolean Operators

```
grunt> a = foreach student generate age,age+10;
```

```
grunt> dump a;
```

```
(20,30)
```

```
(21,31)
```

```
(21,31)
```

```
(22,32)
```

```
(20,30)
```

```
(19,29)
```

```
grunt> A = filter student by age>20;
```

```
grunt> dump A;
```

```
(raju,chennai,21)
(john,bangalore,21)
(sita,hyderabad,22)
```

```
grunt> A = filter student by age>19 and age<22;
grunt> dump A;
```

(similarly, we can use or,not)

```
(ramu,bangalore,20)
(raju,chennai,21)
(john,bangalore,21)
(krish,delhi,20)
```

Problems on module – 2:

Create the following data file

```
[cloudera@localhost ~]$ gedit doctor
```

```
Milan,1001,5,apollo,500
Jay,1002,10,apollo,500
lalit,1003,20,manipal,500
Mohit,1004,15,columbia,600
Chauhan,1005,30,narayana,550
Suraj,1006,25,manipal,650
```

```
[cloudera@localhost ~]$ pig -x local
```

```
grunt> clear
```

```
grunt> doc = load '/home/cloudera/doctor' using PigStorage(',') as  
      (name:chararray, id:int, exp:int, hosp:chararray, fees:int);  
grunt> dump doc;
```

Q) Display the name of the doctor and hospital.

```
grunt> a = foreach doc generate name,hosp;  
grunt> dump a;  
(Milan,apollo)  
(Jay,apollo)  
(lalit,manipal)  
(Mohit, columbia)  
(Chauhan,narayana)  
(Suraj,manipal)
```

Q) Display the details of doctors who have more than 10 years of experience

```
grunt> a = filter doc by exp>10;  
grunt>dump a;  
(lalit,1003,20,manipal,500)  
(Mohit,1004,15, columbia,600)  
(Chauhan,1005,30,narayana,550)  
(Suraj,1006,25,manipal,650)
```

Q)Display the details of doctor who works in apollo hospital.

```
grunt> a = filter doc by hosp=='apollo';
```



```
grunt> dump a;  
(Milan,1001,5,apollo,500)  
(Jay,1002,10,apollo,500)
```

Q)Display doctor name, docid who works in manipal hospital.

```
grunt> a = filter doc by hosp=='manipal';  
grunt> ans = foreach a generate name,id;  
grunt> dump ans;  
(lalit,1003)  
(Suraj,1006)
```

Q) List different fees taken by various doctors.

```
grunt> a = foreach doc generate fees;  
grunt> ans = distinct a;  
grunt> dump ans;  
(500)  
(550)  
(650)  
(600)
```

Q)Display the details of doctor whose experience is more than 20 yrs but not working in manipal.

```
grunt> a = filter doc by exp>20 and hosp!='manipal';  
grunt> dump a;  
(Chauhan,1005,30,narayana,550)
```

(OR)

```
grunt> a = filter doc by exp>20 and not(hosp=='manipal');  
grunt> dump a;  
(Chauhan,1005,30,narayana,550)
```

Q)Display docname, fees, fees%10

```
grunt> a = foreach doc generate name, fees,fees%10;
grunt> dump a;
(Milan,500,0)
(Jay,500,0)
(lalit,500,0)
(Mohit,600,0)
(Chauhan,550,50)
(Suraj,650,50)
```

Q)Display docname,hospname,fees,fees after 10 percentage hike.

```
grunt> a = foreach doc generate name,hosp, fees,fees*1.1;
grunt> dump a;
(Milan,apollo,500,550.0)
(Jay,apollo,500,550.0)
(lalit,manipal,500,550.0)
(Mohit, columbia ,600,660)
(Chauhan,narayana,550,605.0)
(Suraj,manipal,650,715.000000000000001)
```

Q) Display true if experience of doctor is above 10; else display false.

```
grunt> a = foreach doc generate name,exp,(exp>10?'true':'false');
grunt> dump a;
(Milan,5,false)
(Jay,10,false)
(lalit,20,true)
(Mohit,15,true)
(Chauhan,30,true)
(Suraj,25,true)
```

Q)Display only hospital name, fees by taking into consideration that some concession is going on for all branches of apollo hospital by rs 200. So display new fees for apollo hospitals and same current fees for other hospitals.

```
grunt> a = foreach doc generate hosp,(hosp=='apollo'?fees-200:fees);
grunt> dump a;
(apollo,300)
(apollo,300)
(manipal,500)
(columbia,600)
(narayana,550)
(manipal,650)
```

Q)Display the details of doctor who is either taking fees above rs 500 or whose experience is above 20 years.

```
grunt> a = filter doc by fees>500 or exp>20; (OR)
grunt> a = filter doc by (fees>500) or (exp>20); (OR)
grunt> a = filter doc by $4>500 or $2>20;
grunt> dump a;
(Mohit,1004,15,columbia,600)
(Chauhan,1005,30,narayana,550)
(Suraj,1006,25,manipal,650)
```

Q) Display the details of doctor whose name ends with 'an'

```
grunt> a = filter doc by name matches '.*an';
grunt> dump a;
(Milan,1001,5,apollo,500)
```

(Chauhan,1005,30,narayana,550)

Q) Display the details of doctor whose name begins with 'S'

grunt> a = filter doc by name matches 'S.*';

(Suraj,1006,25,manipal,650)

Q) Display the details of doctor whose name has 'a' as substring

grunt> a = filter doc by name matches '.*a.*';

(Milan,1001,5,apollo,500)

(Jay,1002,10,apollo,500)

(lalit,1003,20,manipal,500)

(Chauhan,1005,30,narayana,550)

(Suraj,1006,25,manipal,650)

Q) Display the details of doctor whose name has substring 'it'

grunt> a = filter doc by name matches '.*it.*';

(lalit,1003,20,manipal,500)

(Mohit,1004,15,columbia,600)

Q) Display the details of doctor whose hospital name has substring 'a'

grunt> a = filter doc by hosp matches '.*a.*';

(Milan,1001,5,apollo,500)

(Jay,1002,10,apollo,500)

(lalit,1003,20,manipal,500)

(Mohit,1004,15,columbia,600)

(Chauhan,1005,30,narayana,550)

(Suraj,1006,25,manipal,650)

Q) Perform grouping of tuples wrt hospital name.

grunt> gr = group doc by hosp;

grunt> dump gr;

```
(apollo,{(Milan,1001,5,apollo,500),(Jay,1002,10,apollo,500)})  
(manipal,{(lalit,1003,20,manipal,500),(Suraj,1006,25,manipal,650)})  
(columbia,{(Mohit,1004,15,columbia,600)})  
(narayana,{(Chauhan,1005,30,narayana,550)})
```

Q)Display hospital name, doc names on screen for each hospital.

```
grunt> gr = group doc by hosp;  
grunt> a = foreach gr generate group,doc.name;  
grunt> dump a;  
(apollo,{(Milan),(Jay)})  
(manipal,{(lalit),(Suraj)})  
(columbia,{(Mohit)})  
(narayana,{(Chauhan)})
```

Module – 3

- **Operators:**

- Combining & splitting – UNION, SPLIT
- Sorting – ORDER BY, LIMIT
- Grouping Operator – GROUP, CO-GROUP
- Joining Operator - JOIN(INNER, SELF JOIN)

- **Pig Latin Built-in functions:**

- Eval functions (Avg, Max, Min, Sum, Count, Size, Concat, Tokenize)
- Bag & Tuple Functions
- String Functions
- Math Functions

- **Apache Pig - Running Scripts:**

- Creating pig script
- Commenting pig script
- Executing –running pig script – with/without parameters
- Sample examples

```
[cloudera@localhost ~]$ gedit dr1
```

```
Milan,1001,5,apollo,500  
Jay,1002,10,apollo,500  
lalit,1003,20,manipal,500  
Mohit,1004,15,columbia,600  
Chauhan,1005,30,narayana,550  
Suraj,1006,25,manipal,650
```

```
[cloudera@localhost ~]$ gedit dr2
```

```
meena,2001,20,rxdx,650  
leena,2001,15,st johns,450  
sonam,2002,30,rxdx,600
```

```
[cloudera@localhost ~]$ gedit empy1
```

```
7001,ameena,10,bang  
7002,amit,20,chennai
```

7003,*anand*,30,*bang*
7004,*alen*,15,*hyd*
7005,*alester*,10,*hyd*
7006,*anshul*,5,*Chennai*

[cloudera@localhost ~]\$ gedit pnt1

101,*harinath*,5,*domlur*,1004
102,*nagarjun*,10,*varthur*,1005
103,*chirajeevi*,20,*HAL*,1006
104,*tarun*,25,*HSR*,1004
105,*prabas*,15,*marthahalli*,1006
106,*chaitanya*,30,*belandur*,1003
107,*nani*,27,*krpuram*,1004

[cloudera@localhost ~]\$ pig -x local

grunt> clear

grunt> doc1 = load '/home/cloudera/dr1' using PigStorage(',') as
 (name:chararray, id:int, exp:int, hosp:chararray, fees:int);

grunt> dump doc1;

grunt> doc2 = load '/home/cloudera/dr2' using PigStorage(',') as
 (name:chararray, id:int, exp:int, hosp:chararray, fees:int);

grunt> dump doc2;

UNION OPERATOR

The **UNION** operator of Pig Latin is used to merge the content of two relations. To perform UNION operation on two relations, their columns and domains must be identical.

Syntax

```
grunt> Relation_name3 = UNION Relation_name1, Relation_name2;
```

```
grunt> result = union doc1,doc2;
```

```
grunt> dump result;
```

```
(Milan,1001,5,apollo,500)  
(Jay,1002,10,apollo,500)  
(lalit,1003,20,manipal,500)  
(Mohit,1004,15,columbia,600)  
(Chauhan,1005,30,narayana,550)  
(Suraj,1006,25,manipal,650)  
(,,,)   
(meena,2001,20,rxdx,650)  
(leena,2001,15,st johns,450)  
(sonam,2002,30,rxdx,600)
```

SPLIT OPERATOR

The **SPLIT** operator is used to split a relation into two or more relations.

Syntax

Given below is the syntax of the **SPLIT** operator.

```
grunt> SPLIT Relation1_name INTO Relation2_name IF (condition1), Relation3_name IF(condition2)
```

```
grunt> split doc1 into senior if exp>15,junior if (exp>5 and exp<=15);
```

```
grunt> dump senior;
```

```
(lalit,1003,20,manipal,500)
```

```
(Chauhan,1005,30,narayana,550)
```

```
(Suraj,1006,25,manipal,650)
```

```
grunt> dump junior;
```

```
(Jay,1002,10,apollo,500)
```

```
(Mohit,1004,15,columbia,600)
```

ORDER BY OPERATOR

The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.

Syntax

Given below is the syntax of the **ORDER BY** operator.

```
grunt> Relation_name2 = ORDER Relatin_name1 BY Field (ASC|DESC);
```

```
grunt> a = order doc1 by name asc;
```

```
grunt> dump a;
```

```
(Chauhan,1005,30,narayana,550)
```

```
(Jay,1002,10,apollo,500)
```

```
(Milan,1001,5,apollo,500)
```

```
(Mohit,1004,15,columbia,600)
```

```
(Suraj,1006,25,manipal,650)
```

```
(lalit,1003,20,manipal,500)
```

LIMIT OPERATOR

The **LIMIT** operator is used to get a limited number of tuples from a relation.

Syntax

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

```
grunt> a = limit doc1 2;
```

```
grunt> dump a;
```

```
(Jay,1002,10,apollo,500)
```

```
(Milan,1001,5,apollo,500)
```

GROUP OPERATOR

The **GROUP** operator is used to group the data in a relation. It collects the data having the same key.

Syntax

```
grunt> Group_data = GROUP Relation_name BY age;
```

Q) Display the details of the doctors hospital wise.

```
grunt> gr = group doc1 by hosp;
```

```
grunt> dump gr;
```

```
(apollo,{(Milan,1001,5,apollo,500),(Jay,1002,10,apollo,500)})
```

```
(manipal,{(lalit,1003,20,manipal,500),(Suraj,1006,25,manipal,650)})
```

```
(columbia,{(Mohit,1004,15,columbia,600)})
```

```
(narayana,{(Chauhan,1005,30,narayana,550)})
```

Grouping By Multiple Columns

Q) Display the details of the doctors hospital wise with same fees

```
grunt> a = group doc1 by (hosp,fees);
grunt> dump a;
((apollo,500),{(Milan,1001,5,apollo,500),(Jay,1002,10,apollo,500)})
((manipal,500),{(lalit,1003,20,manipal,500)})
((manipal,650),{(Suraj,1006,25,manipal,650)})
((columbia,600),{(Mohit,1004,15,columbia,600)})
((narayana,550),{(Chauhan,1005,30,narayana,550)})
```

CO-GROUP OPERATOR

The **COGROUP** operator works more or less in the same way as the GROUP operator. The only difference between the two operators is that the **group** operator is normally used with one relation, while the **cogroup** operator is used in statements involving two or more relations.

```
grunt> emp1 = load '/home/cloudera/empy1' using PigStorage(',') as
(id:int, name:chararray, exp:int, place:chararray);
```

```
grunt> a = cogroup doc1 by exp,emp1 by exp;
grunt> dump a;
(5,{(Milan,1001,5,apollo,500)},{(7006,anshul,5,chennai)})
(10,{(Jay,1002,10,apollo,500)},{(7001,ameena,10,bang),(7005,alester,10,hyd)})
(15,{(Mohit,1004,15,columbia,600)},{(7004,alen,15,hyd)})
(20,{(lalit,1003,20,manipal,500)},{(7002,amit,20,chennai)})
(25,{(Suraj,1006,25,manipal,650)},{})
(30,{(Chauhan,1005,30,narayana,550)},{(7003,anand,30,bang)})
```

JOIN OPERATOR

The JOIN operator is used to combine records from two or more relations.

Self-Join

Self-join is used to join a table with itself as if the table were two relations.

```
grunt> doc = load '/home/cloudera/dr1' using PigStorage(',') as  
      (name:chararray, id:int, exp:int, hosp:chararray, fees:int);
```

```
grunt> a = join doc by id, doc1 by id;  
grunt> dump a;  
(Milan,1001,5,apollo,500,Milan,1001,5,apollo,500)  
(Jay,1002,10,apollo,500,Jay,1002,10,apollo,500)  
(lalit,1003,20,manipal,500,lalit,1003,20,manipal,500)  
(Mohit,1004,15,columbia,600,Mohit,1004,15,columbia,600)  
(Chauhan,1005,30,narayana,550,Chauhan,1005,30,narayana,550)  
(Suraj,1006,25,manipal,650,Suraj,1006,25,manipal,650)
```

Inner-Join

It is also referred to as equijoin. An inner join returns rows when there is a match in both tables.

```
grunt> pat = load '/home/cloudera/pnt1' using PigStorage(',') as (pid:int,name:chararray,age:int,addr:chararray,docid:int);
```

Q)display entire details of patient and their corresponding doctor

```
grunt> a = join doc1 by id, pat by docid;  
grunt> dump a;  
(lalit,1003,20,manipal,500,106,chaitanya,30,belandur,1003)  
(Mohit,1004,15,columbia,600,101,harinath,5,domlur,1004)  
(Mohit,1004,15,columbia,600,104,tarun,25,HSR,1004)  
(Mohit,1004,15,columbia,600,107,nani,27,krpuram,1004)  
(Chauhan,1005,30,narayana,550,102,nagarjun,10,varthur,1005)
```

(Suraj,1006,25,manipal,650,103,chirajeevi,20,HAL,1006)
(Suraj,1006,25,manipal,650,105,prabas,15,marthahalli,1006)

Pig Latin Built-in functions:

Eval functions (Avg, Max, Min, Sum, Count, Size, Concat, Tokenize)

```
[cloudera@localhost ~]$ gedit dr1
```

Milan,1001,5,apollo,500

Jay,1002,10,apollo,500

lalit,1003,20,manipal,500

Mohit,1004,15,columbia,600

Chauhan,1005,30,narayana,550

Suraj,1006,25,manipal,650

```
[cloudera@localhost ~]$ pig -x local
```

```
grunt> clear
```

```
grunt> doc = load '/home/cloudera/dr1' using PigStorage(',') as
```

```
    (name:chararray, id:int, exp:int, hosp:chararray, fees:int);
```

```
grunt> dump doc;
```

(Milan,1001,5,apollo,500)

(Jay,1002,10,apollo,500)

(lalit,1003,20,manipal,500)

(Mohit,1004,15,columbia,600)

(Chauhan,1005,30,narayana,550)
(Suraj,1006,25,manipal,650)

Group All

You can group a relation by all the columns as shown below.

```
grunt> group_all = GROUP relation_name All;
```

```
grunt> gr = group doc all;
```

```
grunt> dump gr;
```

(all,{(Milan,1001,5,apollo,500),(Jay,1002,10,Apollo,500),(lalit,1003,20,manipal,500),(Mohit,15,1004,15,)),(Chauhan,1005,30,narayana,550),(Suraj,1006,25,manipal,650),(Jay,102,10,apollo,50)})

AVG():To compute the average of the numerical values within a bag.

Q)Display hospital name, fees and average fees among all the hospital.

```
grunt> result = foreach gr generate doc.hosp,doc.fees,AVG(doc.fees);
```

({(apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},550.0)

MAX():To calculate the highest value for a column in a single-column bag.

Q)Display hospital name, fees and maximum fees among all the hospital.

```
grunt> result = foreach gr generate doc.hosp,doc.fees,MAX(doc.fees);
```

```
grunt> dump result;
```

({(apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},650)

MIN(): To get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag.

Q)Display hospital name, fees and minimum fees among all the hospital.

```
grunt> result = foreach gr generate doc.hosp,doc.fees,MIN(doc.fees);
```

```
grunt> dump result;
```

```
(({apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},{500})
```

SUM(): To get the total of the numeric values of a column in a single-column bag.

Q)Display hospital name, fees and total fees among all the hospital.

```
grunt> result = foreach gr generate doc.hosp,doc.fees,SUM(doc.fees);
```

```
grunt> dump result;
```

```
(({apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},{3300})
```

COUNT(): To get the the number of tuples in a bag.

Q)Display total no:of tuples/rows in relation.

```
grunt> result = foreach gr generate COUNT(doc.id);
```

```
grunt> dump result;
```

```
(6)
```

SIZE(): To compute the number of elements based on any Pig data type.

Q)Display doctor name along with the length of doctor name in each row.

```
grunt> ans = foreach doc generate name,SIZE(name);
```



```
grunt> dump ans;
```

```
(Milan,5)
```

```
(Jay,3)
```

```
(lalit,5)
```

```
(Mohit,5)
```

```
(Chauhan,7)
```

```
(Suraj,5)
```

CONCAT(): To concatenate two or more expressions of same type.

```
grunt> ans = foreach doc generate CONCAT(name,hosp);
```

```
grunt> dump ans;
```

```
(Milanapollo)
```

```
(Jayapollo)
```

```
(lalitmanipal)
```

```
(Mohitcolumbia)
```

```
(Chauhannarayana)
```

```
(Surajmanipal)
```

Bag & Tuple Functions

TUPLE CONSTRUCTION:

```
grunt> a = foreach doc generate name,id,exp;
```

```
grunt> dump a;
```

(Milan,1001,5)
(Jay,1002,10)
(lalit,1003,20)
(Mohit,1004,15)
(Chauhan,1005,30)
(Suraj,1006,25)

BAG CONSTRUCTION:

```
grunt> a = foreach doc generate {(name,id,exp)},{name,id,exp};
```

```
grunt> dump a;
```

{(Milan,1001,5)}, {(Milan),(1001),(5)}
{(Jay,1002,10)}, {(Jay),(1002),(10)}
{(lalit,1003,20)}, {(lalit),(1003),(20)}
{(Mohit,1004,15)}, {(Mohit),(1004),(15)}
{(Chauhan,1005,30)}, {(Chauhan),(1005),(30)}
{(Suraj,1006,25)}, {(Suraj),(1006),(25)}

MAP CONSTRUCTION:

```
grunt> a = foreach doc generate [name,exp];
```

```
grunt> dump a;
```

[Milan#5]
[Jay#10]
[lalit#20]
[Mohit#15]
[Chauhan#30]
[Suraj#25]

STRING BUILT IN FUNCTIONS

SUBSTRING()

Returns a substring from a given string.

Syntax:

SUBSTRING(string, startIndex, ending index+1)

```
grunt> ans = foreach doc generate (id,name),SUBSTRING (name, 0 , 2);
grunt> dump ans;
((1001,Milan),Mi)
((1002,Jay),Ja)
((1003,lalit),la)
((1004,Mohit),Mo)
((1005,Chauhan),Ch)
((1006,Suraj),Su)
```

INDEXOF():Returns the first occurrence of a character in a string, searching forward from a start index.

Syntax:

INDEXOF(string, 'character', startIndex)

```
grunt> ans = foreach doc generate (id,name),INDEXOF(name,'a',0);
grunt> dump ans;
((1001,Milan),3)
((1002,Jay),1)
((1003,lalit),1)
((1004,Mohit),-1)
((1005,Chauhan),2)
((1006,Suraj),3)
```

LCFIRST(): Converts the first character in a string to lower case.

Syntax:

LCFIRST(expression)

```
grunt> ans = foreach doc generate (id,name),LCFIRST(name);
```

```
grunt> dump ans;
```

```
((1001,Milan),milan)
```

```
((1002,Jay),jay)
```

```
((1003,lalit),lalit)
```

```
((1004,Mohit),mohit)
```

```
((1005,Chauhan),chauhan)
```

```
((1006,Suraj),suraj)
```

UCFIRST(): Returns a string with the first character converted to upper case.

Syntax:

UCFIRST(expression)

```
grunt> ans = foreach doc generate (id,hosp),UCFIRST(hosp);
```

```
grunt> dump ans;
```

```
((1001,apollo),Apollo)
```

```
((1002,apollo),Apollo)
```

```
((1003,manipal),Manipal)
```

```
((1004,columbia),Columbia)
```

```
((1005,narayana),Narayana)
```

```
((1006,manipal),Manipal)
```

UPPER():Returns a string converted to upper case

Syntax:

UPPER(expression)

```
grunt> ans = foreach doc generate (id,name),UPPER(name);
grunt> dump ans;
((1001,Milan),MILAN)
((1002,Jay),JAY)
((1003,lalit),LALIT)
((1004,Mohit),MOHIT)
((1005,Chauhan),CHAUHAN)
((1006,Suraj),SURAJ)
```

LOWER(): Converts all characters in a string to lower case.

Syntax:

LOWER(expression)

```
grunt> ans = foreach doc generate (id,name),LOWER(name);
grunt> dump ans;
((1001,Milan),milan)
((1002,Jay),jay)
((1003,lalit),lalit)
((1004,Mohit),mohit)
((1005,Chauhan),chauhan)
((1006,Suraj),suraj)
```

REPLACE(): To replace existing characters in a string with new characters.

Syntax:

REPLACE(string, 'oldChar', 'newChar');

```
grunt> ans = foreach doc generate (id,hosp),REPLACE(hosp,'apollo','appo');
grunt> dump ans;
((1001,apollo),appo)
((1002,apollo),appo)
((1003,manipal),manipal)
((1004,columbia),columbia)
((1005,narayana),narayana)
((1006,manipal),manipal)
```

BUILT_IN MATH FUNCTIONS

\$gedit math1.txt

```
5
16
9
2.5
2
3.5
3.14
-2.2
```

```
grunt> mat = load '/home/cloudera/math1.txt' using PigStorage(',') as
(data:float);
```

ABS(): ABSOLUTE VALUE

To get the absolute value of an expression

```
grunt> ans = foreach mat generate data,ABS(data);
grunt> dump ans;
(5.0,5.0)
(16.0,16.0)
```

(9.0,9.0)
(2.5,2.5)
(2.0,2.0)
(3.5,3.5)
(3.14,3.14)
(-2.2,2.2)

CBRT() : cube root

This function is used to get the cube root of an expression.

```
grunt> ans = foreach mat generate data,CBRT(data);
```

```
grunt> dump ans;
```

(5.0,1.709975946676697)
(16.0,2.5198420997897464)
(9.0,2.080083823051904)
(2.5,1.3572088082974532)
(2.0,1.2599210498948732)
(3.5,1.5182944859378313)
(3.14,1.464344366810533)
(-2.2,-1.300591456247907)

SBRT() : square root

To get the positive square root of an expression.

```
grunt> ans = foreach mat generate data,SQRT(data);
```

```
grunt> dump ans;
```

(5.0,2.23606797749979)
(16.0,4.0)
(9.0,3.0)
(2.5,1.5811388300841898)
(2.0,1.4142135623730951)

(3.5,1.8708286933869707)
(3.14,1.7720045442673602)
(-2.2,NaN)

COS():

This function is used to get the trigonometric cosine of an expression.

```
grunt> ans = foreach mat generate data,COS(data);
```

```
grunt> dump ans;
```

(5.0,0.28366218546322625)
(16.0,-0.9576594803233847)
(9.0,-0.9111302618846769)
(2.5,-0.8011436155469337)
(2.0,-0.4161468365471424)
(3.5,-0.9364566872907963)
(3.14,-0.99999873189461)
(-2.2,-0.5885011558074578)

SIN():

To get the sine of an expression.

```
grunt> ans = foreach mat generate data,SIN(data);
```

```
grunt> dump ans;
```

(5.0,-0.9589242746631385)
(16.0,-0.2879033166650653)
(9.0,0.4121184852417566)
(2.5,0.5984721441039564)
(2.0,0.9092974268256817)
(3.5,-0.35078322768961984)
(3.14,0.0015925480124451862)
(-2.2,-0.8084963757576692)

TAN():

To get the trigonometric tangent of an angle.

```
grunt> ans = foreach mat generate data,TAN(data);
```

```
grunt> dump ans;
```

```
(5.0,-3.380515006246586)  
(16.0,0.3006322420239034)  
(9.0,-0.45231565944180985)  
(2.5,-0.7470222972386603)  
(2.0,-2.185039863261519)  
(3.5,0.3745856401585947)  
(3.14,-0.0015925500319664656)  
(-2.2,1.37382291908733)
```

CEIL():

This function is used to get the value of an expression rounded up to the nearest integer.

```
grunt> ans = foreach mat generate data,CEIL(data);
```

```
grunt> dump ans;
```

```
(5.0,5.0)  
(16.0,16.0)  
(9.0,9.0)  
(2.5,3.0)  
(2.0,2.0)  
(3.5,4.0)  
(3.14,4.0)  
(-2.2,-2.0)
```

FLOOR():

To get the value of an expression rounded down to the nearest integer.

```
grunt> ans = foreach mat generate data,FLOOR(data);
```

```
grunt> dump ans;
```

```
(5.0,5.0)
```

```
(16.0,16.0)
```

```
(9.0,9.0)
```

```
(2.5,2.0)
```

```
(2.0,2.0)
```

```
(3.5,3.0)
```

```
(3.14,3.0)
```

```
(-2.2,-3.0)
```

ROUND():

To get the value of an expression rounded to an integer (if the result type is float) or rounded to a long (if the result type is double).

```
grunt> ans = foreach mat generate data,ROUND(data);
```

```
grunt> dump ans;
```

```
(5.0,5)
```

```
(16.0,16)
```

```
(9.0,9)
```

```
(2.5,3)
```

```
(2.0,2)
```

```
(3.5,4)
```

```
(3.14,3)
```

```
(-2.2,-2)
```

EXP():

This function is used to get the Euler's number e raised to the power of x.

```
grunt> ans = foreach mat generate data, EXP(data);
```

```
grunt> dump ans;
```

```
(5.0,148.4131591025766)
```

```
(16.0,8886110.520507872)
```

```
(9.0,8103.083927575384)
```

```
(2.5,12.182493960703473)
```

```
(2.0,7.38905609893065)
```

```
(3.5,33.11545195869231)
```

```
(3.14,23.103869282414397)
```

```
(-2.2,0.1108031530788277)
```

LOG10():

To get the base 10 logarithm of an expression.

```
grunt> ans = foreach mat generate data, LOG10(data);
```

```
grunt> dump ans;
```

```
(5.0,0.6989700043360189)
```

```
(16.0,1.2041199826559248)
```

```
(9.0,0.9542425094393249)
```

```
(2.5,0.3979400086720376)
```

```
(2.0,0.3010299956639812)
```

```
(3.5,0.5440680443502757)
```

```
(3.14,0.4969296625825472)
```

```
(-2.2,NaN)
```

LOG():

To get the natural logarithm (base e) of an expression.

```
grunt> ans = foreach mat generate data, LOG(data);
```

```
grunt> dump ans;
```

```
(5.0,1.6094379124341003)
```

(16.0,2.772588722239781)
(9.0,2.1972245773362196)
(2.5,0.9162907318741551)
(2.0,0.6931471805599453)
(3.5,1.252762968495368)
(3.14,1.1442228333291342)
(-2.2,NaN)

Apache Pig - Running Scripts

Word count Program:

To create a data file

```
[cloudera@localhost ~]$ gedit datafile7
```

```
hai hello how are you  
are you fine  
the world is very beautiful  
my name is raj  
i live in bangalore  
i am fine
```

Save & Close

To put this file into hadoop

```
[cloudera@localhost ~]$ hadoop fs -put datafile7 /user/cloudera/
```

To create a pig latin script

```
[cloudera@localhost ~]$ gedit wcount7.pig
```

```
file = load '/user/cloudera/datafile7' as (line:chararray);  
t = foreach file generate flatten(TOKENIZE(line)) as word;  
gr = group t by word;  
wc = foreach gr generate group,COUNT(t.word);  
dump wc;  
store wc into '/user/cloudera/script_o7' using PigStorage();
```

Save & Close

To run the pig latin script

```
[cloudera@localhost ~]$ pig wcount7.pig
```

```
2018-03-28 00:24:21,302 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

To check the output

```
[cloudera@localhost ~]$ hadoop fs -ls /user/cloudera/script_o7
```

Found 3 items

```
-rw-r--r--  3 cloudera cloudera    0 2018-03-28 00:24 /user/cloudera/script_o7/_SUCCESS
drwxr-xr-x  - cloudera cloudera    0 2018-03-28 00:23 /user/cloudera/script_o7/_logs
-rw-r--r--  3 cloudera cloudera 128 2018-03-28 00:24 /user/cloudera/script_o7/part-r-00000
```

```
[cloudera@localhost ~]$ hadoop fs -cat /user/cloudera/script_o7/part*
```

```
i      2
am     1
in     1
is     2
my     1
are    2
hai    1
how    1
raj    1
the    1
you    2
fine   2
live   1
name   1
very   1
hello  1
world  1
bangalore    1
beautiful    1
```

Grep Program:

To create a pig latin script

```
[cloudera@localhost ~]$ gedit grep.pig
```

```
file = load '/user/cloudera/datafile1' as (line:chararray);  
token = foreach file generate flatten(TOKENIZE(line)) as word;  
t = filter token by word matches 'you';  
gr = group t by word;  
wc = foreach gr generate group,COUNT(t.word);  
dump wc;  
store wc into '/user/cloudera/script_o8' using PigStorage();
```

Save & Close

To run the pig latin script

```
[cloudera@localhost ~]$ pig grep.pig
```

```
2018-03-28 00:43:05,939 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

To check the output

```
[cloudera@localhost ~]$ hadoop fs -ls /user/cloudera/script_o2
```

Found 3 items

```
-rw-r--r--  3 cloudera cloudera    0 2018-03-28 00:43 /user/cloudera/script_o8/_SUCCESS  
drwxr-xr-x  - cloudera cloudera    0 2018-03-28 00:42 /user/cloudera/script_o8/_logs  
-rw-r--r--  3 cloudera cloudera    6 2018-03-28 00:42 /user/cloudera/script_o8/part-r-00000
```

```
[cloudera@localhost ~]$ hadoop fs -cat /user/cloudera/script_o8/part*
```

Module – 4 (HIVE)

What is Hive?

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive.

Hive primitive data types: tinyint, smallint, int, bigint, string etc.

To connect to hive

```
[cloudera@localhost ~]$ hive
```

TO SWITCH OFF SAFE MODE

```
$ sudo -u hdfs hdfs dfsadmin -safemode leave
```

DDL Commands

To Create Two Files On Local Filesystem And Copy It To Hdfs Any Folder

```
[cloudera@localhost ~]$ gedit emp.txt
```

```
1001 | hari | d1 | chennai | 1986-12-10
```

```
1002 | teja | d1 | hyd | 1987-01-21
```

```
1003 | ram | d3 | delhi | 1986-02-11
```

```
1004 | milind | d4 | bang | 1988-03-21
```

```
1005 | jay | d2 | bang | 1988-03-22
```

```
1006 | naveen | d4 | hyd | 1986-04-12
```

```
1007 | naser | d1 | hyd | 1989-11-15
```

```
1008 | rahul | d3 | delhi | 1990-12-23
```



```
[cloudera@localhost ~]$ gedit d.txt
```

```
d1 | research | A-block
```

```
d2 | sales | A-block
```

```
d3 | testing | B-block
```

```
d4 | development | C-block
```

```
[cloudera@localhost ~]$ hadoop fs -put emp.txt /user/cloudera/batch3
```

```
[cloudera@localhost ~]$ hadoop fs -put d.txt /user/cloudera/batch3
```

CONNECT TO HIVE

```
[cloudera@localhost ~]$ hive
```

```
hive>
```

TO CREATE DATABASE

```
hive> ok; (OR)
```

```
hive> create database if not exists test;
```

TO LIST OUT DATABASES

```
hive> show databases;
```

TO DROP DATABASE

```
hive> drop database test; (OR)
```

```
hive> drop database if exists test; (OR)
```

```
hive> drop database if exists test cascade;
```

NOTE!!!! [if exists] & [if not exists] doesn't show error if database already exists while creating time and database doesn't exists while dropping the same.

Without these options , errors displayed clearly.

TO MAKE USE OF THE DATABASE

```
hive> use test;
```

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

TO CREATE TABLE

```
hive> create table emp(id int,name string,dept string,place string,dob string)
>comment 'this is employee table'
> row format delimited fields terminated by '|' lines terminated by '\n'
>stored as textfile;
```

(OR) Type IN Single Line

```
hive> create table emp(id int,name string,dept string,place string,dob string) comment 'this is employee table'
row format delimited fields terminated by '|' lines terminated by '\n' stored as textfile;
```

```
hive> create table department(did string,dname string,block string) comment 'this is department table' row format
delimited fields terminated by '|' lines terminated by '\n' stored as textfile;
```

NOTE!!! You can mention just mention

```
Hive>USE test;
```

```
Hive> CREATE TABLE emp (.....) ....
```

(OR)

```
Hive> CREATE TABLE test.emp (.....) ....
```

TO SEE THE LIST OF TABLES

```
hive> show tables;
```

TO SEE THE STRUCTURE OF A TABLE

```
hive> describe emp;
```

TO SEE THE STRUCTURE & METADATA INFORMATION OF TABLE

```
hive> describe formatted emp;
```

```
hive> show create table emp;
```

Contents of directory [/user/hive/warehouse/test.db](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
department	dir				2018-04-09 23:38	rw-rw-rw-	cloudera	hive
emp	dir				2018-04-09 23:57	rw-rw-rw-	cloudera	hive

Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

```
ALTER TABLE name RENAME TO new_name
```

```
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
```

```
ALTER TABLE name CHANGE column_name new_name new_type
```

ALTER TABLE name REPLACE COLUMNS (col_spec [, col_spec ...])

TO RENAME TABLE NAME

hive> alter table department rename to d;

hive> show tables;

d

emp

TO ADD ONE OR MORE COLUMNS TO THE TABLE

hive> alter table d add columns (estb_year int, rating smallint);

hive> describe d;

did string

dname string

block string

estb_year int

rating smallint

TO CHANGE COLUMN NAME OR ITS DATATYPE OR BOTH

hive> alter table d change rating rate string;

hive> describe d;

did string

dname string

block string

estb_year int

rate string

hive> alter table d change rate rate bigint;

hive> describe d;

did string

dname string

```
block string
estb_year int
rate bigint
```

TO REPLACE COLUMNS

```
hive> alter table d replace columns (did string,dname string, block string);
```

```
hive> describe d;
```

```
did string
dname string
block string
```

```
hive> alter table d replace columns (block string);
```

```
hive> describe d;
```

```
block string
```

```
hive> select * from d;
```

```
d1
d2
d3
d4
```

//if you do REPLACE again, you will get the columns again I,e replace is not removing columns permanently

```
hive> alter table d replace columns (did string,dname string, block string);
```

```
hive> desc d;
```

```
did string
dname string
block string
```

```
hive> select * from d;
```

d1	research	A-block
d2	sales	A-block
d3	testing	B-block
d4	development	C-block
d5	hr	A-block

TO DROP THE TABLE

hive> drop table if exists d; (OR)
hive> drop table d;

Load, Insert of data

Load Data Statement

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

Syntax

The syntax for load data is as follows:

**LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]**

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional used table is created with partitions.

TO LOAD FROM LOCAL FILESYSTEM

hive> load data local inpath '/home/cloudera/emp.txt' into table emp;
hive> select * from emp;

```

1001 hari d1 chennai 1986-12-10
1002 teja d1 hyd 1987-01-21
1003 ram d3 delhi 1986-02-11
1004 milind d4 bang 1988-03-21
1005 jay d2 bang 1988-03-22
1006 naveen d4 hyd 1986-04-12
1007 naser d1 hyd 1989-11-15
1008 rahul d3 delhi 1990-12-23

```

Contents of directory [/user/hive/warehouse/test.db/emp](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
emp.txt	file	238 B	3	128 MB	2018-04-09 23:57	rw-rw-rw-	cloudera	supergroup

TO LOAD FROM HADOOP FILE SYSTEM

hive> load data inpath '/user/cloudera/emp.txt' into table emp;

hive> select * from emp;

```

1001 hari d1 chennai 1986-12-10
1002 teja d1 hyd 1987-01-21
1003 ram d3 delhi 1986-02-11
1004 milind d4 bang 1988-03-21
1005 jay d2 bang 1988-03-22
1006 naveen d4 hyd 1986-04-12
1007 naser d1 hyd 1989-11-15
1008 rahul d3 delhi 1990-12-23
1001 hari d1 chennai 1986-12-10

```

```

1002 teja d1 hyd 1987-01-21
1003 ram d3 delhi 1986-02-11
1004 milind d4 bang 1988-03-21
1005 jay d2 bang 1988-03-22
1006 naveen d4 hyd 1986-04-12
1007 naser d1 hyd 1989-11-15
1008 rahul d3 delhi 1990-12-23

```

Contents of directory [/user/hive/warehouse/test.db/emp](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
emp.txt	file	238 B	3	128 MB	2018-04-09 23:48	rw-rw-rw-	cloudera	supergroup
emp_copy_1.txt	file	238 B	3	128 MB	2018-04-09 22:25	rw-rw-rw-	cloudera	cloudera

NOTE!!!!Here, Since no overwrite was used; the data got appended to same table. And in Hive/warehouse/test.db two copies of same content got generated.

TO LOAD USING OVERWRITE KEYWORD

```
hive> load data local inpath '/home/cloudera/emp.txt' overwrite into table emp;
```

```
hive> select * from emp;
```

```

1001 hari d1 chennai NULL
1002 teja d1 hyd NULL
1003 ram d3 delhi NULL
1004 milind d4 bang NULL
1005 jay d2 bang NULL
1006 naveen d4 hyd NULL
1007 naser d1 hyd NULL
1008 rahul d3 delhi NULL

```


Contents of directory [/user/hive/warehouse/test.db/emp](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
emp.txt	file	238 B	3	128 MB	2018-04-10 00:19	rw-rw-rw-	cloudera	supergroup

NOTE!!! Once the data is loaded to hive table from hadoop filesystem, the file “emp.txt” no more exists in /user/cloudera I,e loading from hadoop filesystem is like cut and paste to hive; whereas its like copy & paste when loaded from local filesystem.

So , if you have loaded from hadoop filesystem once, then you can't load or load with overwrite to hive table from hadoop filesystem again.... Because you will get ERROR: “invalid path as file is cut already from that location.”

```
hive> load data local inpath '/home/cloudera/d.txt' overwrite into table department;
```

```
hive> select * from department;
```

```
d1    research    A-block
```

```
d2    sales    A-block
```

```
d3    testingB-block
```

```
d4    development C-block
```

Contents of directory [/user/hive/warehouse/test.db/department](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
d.txt	file	79 B	3	128 MB	2018-04-10 03:00	rw-rw-rw-	cloudera	supergroup

BUILT IN RELATIONAL OPERATORS

A=B, A<B, A<=B, A>B, A>=B, A IS NULL, A IS NOT NULL, A LIKE B, A!=B

BUILT IN ARTHMETIC OPERATORS

A+B, A-B, A*B, A/B, A%B, A&B, A|B, A^B, ~A

A&B : bitwise and operation

A|B: bitwise or operation

A^B;bitwise XOR operation

~A: bitwise not operation

BUILT IN LOGICAL OPERATORS

A AND B, A OR B, NOT A, A || B, A&&B, !A

Q)Display details of employee whose employee id is greater than and equal to 1003 and doesn't come from hyd.

hive> select * from emp where id >=1003 and place !='hyd'; (OR)

hive> select * from emp where id >=1003 and place not in ('hyd');

1003 ram d3 delhi 1986-02-11

1004 milindd4 bang 1988-03-21

1005 jay d2 bang 1988-03-22

1008 rahul d3 delhi 1990-12-23

Q)Display details of department whose id is less than d2 or department name is development.

hive> select * from department where did<'d2' or dname='development'; (OR)

hive> select * from department where did<'d2' or dname like 'development'; (OR)

hive> select * from department where did<'d2' or dname like 'd%';

d1 research A-block

d4 development C-block

Q)Display details of department whose department name's second letter is 'e'.

hive> select * from department where did<'d2' or dname like '_e%';

d1 research A-block

d3 testingB-block

d4 development C-block

Q)Display total no:of employees ,minimum of their employee no, avg of their employee no,max of their employee, sum of their employee from employee dataset.

hive> select count(*),min(id),avg(id),max(id),sum(id) from emp;

8 1001 1004.5 1008 8036

Q) Display department id, count of employees in each department

hive> select count(*),dept from emp group by dept;

3 d1

1 d2

2 d3

2 d4

Q) Display department id,count of employees in each department and display rows those have count more than 2.

hive> select count(*),dept from emp group by dept having count(*)>2;

3 d1

Q) Display department id,count of employees in each department in descending order of count.

hive> select count(*) as c,dept from emp group by dept order by c desc;

3 d1

2 d3

2 d4

1 d2

Q) Display department id,count of employees in each department in descending order of count and display only first two rows.

hive> select count(*) as c,dept from emp group by dept order by c desc limit 2;

3 d1

2 d3

TO JOIN TWO TABLES

hive> select * from emp e join department d on (e.dept=d.did);

001 hari d1 chennai 1986-12-10 d1 research A-block

1002 teja d1 hyd 1987-01-21 d1 research A-block

1007 naser d1 hyd 1989-11-15 d1 research A-block

1005 jay d2 bang 1988-03-22 d2 sales A-block

1003 ram d3 delhi 1986-02-11 d3 testing B-block

1008 rahul d3 delhi 1990-12-23 d3 testing B-block

1004 milind d4 bang 1988-03-21 d4 development C-block

1006 naveen d4 hyd 1986-04-12 d4 development C-block

hive> select * from emp e left outer join department d on (e.dept=d.did);

1001 hari d1 chennai 1986-12-10 d1 research A-block

1002 teja d1 hyd 1987-01-21 d1 research A-block

1007 naser d1 hyd 1989-11-15 d1 research A-block

1005 jay d2 bang 1988-03-22 d2 sales A-block

1003 ram d3 delhi 1986-02-11 d3 testing B-block

1008 rahul d3 delhi 1990-12-23 d3 testing B-block

1004 milind d4 bang 1988-03-21 d4 development C-block

1006 naveen d4 hyd 1986-04-12 d4 development C-block

1009 jay d6 hyd 1988-07-19 null null null

NOTE!!if d6 department not there ,then no matching on right side table values

```
hive> select * from emp e right outer join department d on (e.dept=d.did);
```

1001	hari	d1	chennai	1986-12-10	d1	research	A-block
1002	teja	d1	hyd	1987-01-21	d1	research	A-block
1007	naser	d1	hyd	1989-11-15	d1	research	A-block
1005	jay	d2	bang	1988-03-22	d2	sales	A-block
1003	ram	d3	delhi	1986-02-11	d3	testing	B-block
1008	rahul	d3	delhi	1990-12-23	d3	testing	B-block
1004	milind	d4	bang	1988-03-21	d4	development	C-block
1006	naveen	d4	hyd	1986-04-12	d4	development	C-block
	NULL	NULL	NULL	NULL	d5	hr	A-block

TO CREATE A VIEW

```
hive> create view emp_v as select id,name from emp where id>1003;
```

```
hive> select * from emp_v;
```

1004	milind
1005	jay
1006	naveen
1007	naser
1008	rahul

TO DROP THE VIEW

```
hive> drop view emp_v;
```

To Use Built-In Functions

```
hive> select upper(name) from emp;
```

HARI
TEJA
RAM

MILIND
JAY
NAVEEN
NASER
RAHUL

hive> select count(id) from emp;
8

hive> select substr(name,1,3) from emp;
har
tej
ram
mil
jay
nav
nas
rah

hive> select substr(name,2) from emp;
ari
eja
am
ilind
ay
aveen
aser
ahul

hive> select substr(name,3,2) from emp;

ri
ja
m
li
y
ve
se
hu

Syntax: substr(string,starting index,no of character)

Note: if no of characters not mentioned then it returns from the start position to the end of the string

TO CREATE A TABLE FROM ANOTHER TABLE

hive> create table abc as select * from emp;

hive> select * from abc;

TO STORE THE OUTPUT OF ANALYSIS TO SOME OTHER TABLE

hive> insert overwrite table abc select * from emp where id>1003;

hive> select * from abc;

004	Milind	D4	Bang	1988-03-21
1005	Jay	D2	Bang	1988-03-22
1006	Naveen	D4	Hyd	1986-04-12
1007	Naser	D1	Hyd	1989-11-15
1008	Rahul	D3	Delhi	1990-12-23

Note:schema should match(No.of column should match)

To Store The Output Of Analysis To Hdfs File System

Hive> Insert Overwrite Directory '/User/Cloudera/Output1' Select * From Emp Where Id>1003;

Note!! Where Output1 Is A New Directory In /User/Cloudera, Which Will Get Created Automatically

To Check The Output File In Hadoop File System

```
[Cloudera@Localhost ~]$ Hadoop Fs -Ls /User/Cloudera/Output1
```

```
[Cloudera@Localhost ~]$ Hadoop Fs -Cat /User/Cloudera/Output1/000000_0
```

(Or)

```
Hive> Dfs -Ls /User/Cloudera/Output1;
```

```
-Rw-R--R-- 3 Cloudera Supergroup      149 2018-04-17 03:32 /User/Cloudera/Output1/000000_0
```

```
Hive> Dfs -Cat /User/Cloudera/Output1/0*;
```

```
1004milindd4bang1988-03-21
```

```
1005jayd2bang1988-03-22
```

```
1006naveend4hyd1986-04-12
```

```
1007naserd1hyd1989-11-15
```

```
1008rahuld3delhi1990-12-23
```

(Or)

Use Browser Of Your Vm To Go To The Location And Verify The File

File: [/user/cloudera/output1/000000_0](#)

Goto :

[Go back to dir listing](#)
[Advanced view/download options](#)

```
1004milindd4bang1988-03-21
1005jayd2bang1988-03-22
1006naveend4hyd1986-04-12
1007naserd1hyd1989-11-15
1008rahuld3delhi1990-12-23
```

TO STORE THE OUTPUT OF ANALYSIS TO LOCAL FILE SYSTEM

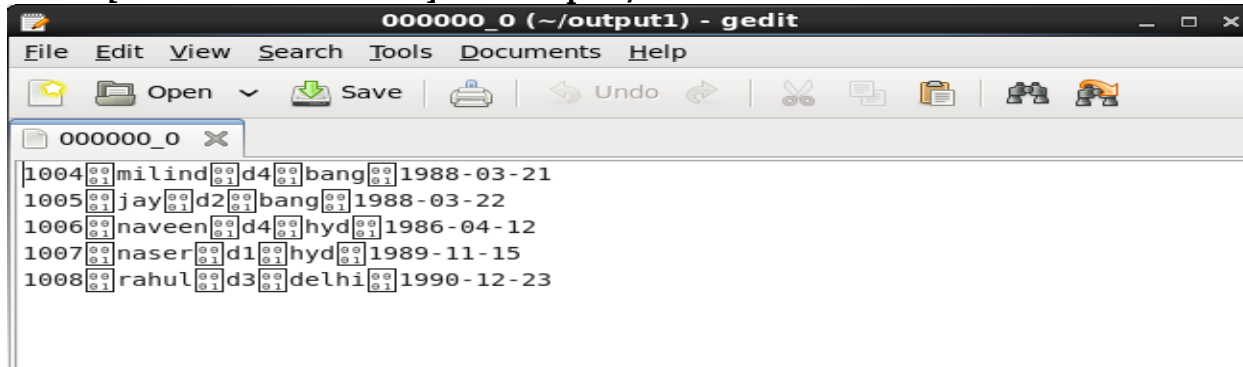
```
hive> insert overwrite local directory '/home/cloudera/output1' select * from employee where id>1003;
```


TO CHECK THE OUTPUT FILE IN LOCAL SYSTEM

- o [cloudera@localhost ~]\$ ls
[cloudera@localhost ~]\$cd output1
[cloudera@localhost ~]\$ cat 000000_0 (OR) \$gedit 000000_0

(OR)

- o [cloudera@localhost ~]\$ ls output1
[cloudera@localhost ~]\$ cat output1/000000_0



```
000000_0 (~/.output1) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
1004 milind d4 bang 1988-03-21
1005 jay d2 bang 1988-03-22
1006 naveen d4 hyd 1986-04-12
1007 naser d1 hyd 1989-11-15
1008 rahul d3 delhi 1990-12-23
```

TO QUIT FROM HIVE

hive> quit;

HBASE (Module – 5)

- HBase is a distributed column-oriented database built on top of the Hadoop file system.
- HBase sits on top of the Hadoop File System and provides read and write access.
- HBase is used whenever we need to provide fast random access to available data.

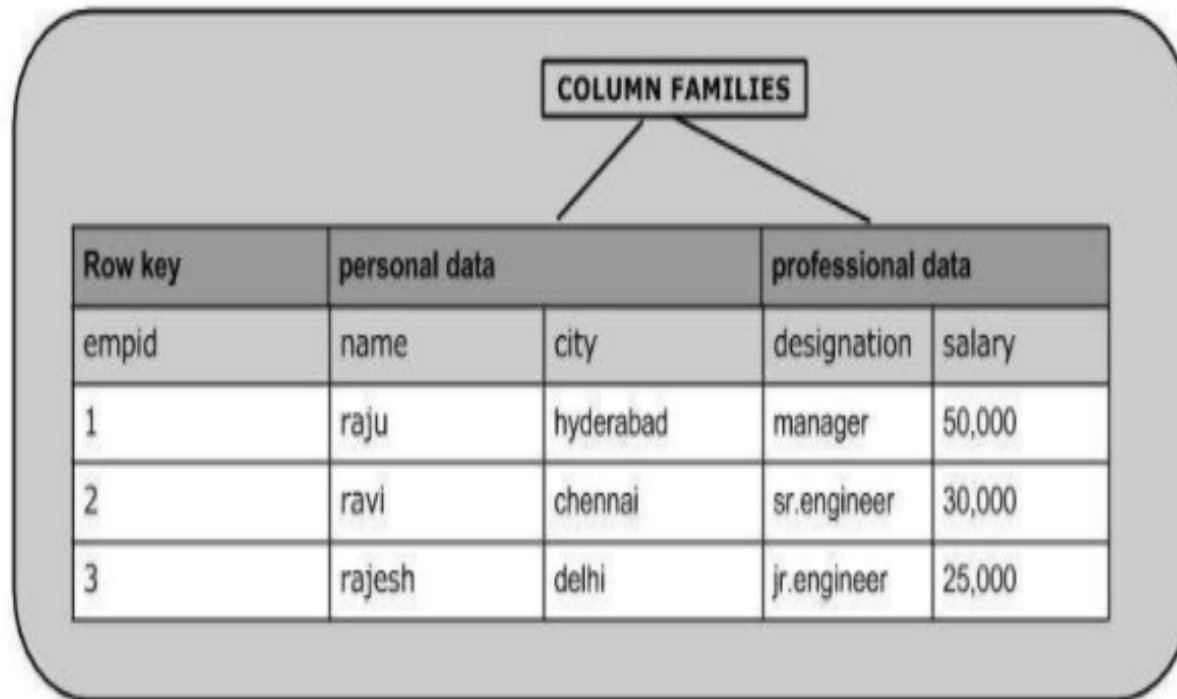
Storage Mechanism in HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Given below is an example schema of table in HBase.

[illegible]

The following image shows column families in a column-oriented database:



TO START HBASE

```
[cloudera@localhost ~]$ hbase shell
```

GENERAL SHELL COMMANDS

hbase(main):001:0> status

1 servers, 0 dead, 2.0000 average load

hbase(main):002:0> version

0.94.6-cdh4.4.0, rUnknown, Tue Sep 3 20:09:51 PDT 2013

hbase(main):003:0> whoami

cloudera (auth:SIMPLE)

hbase(main):004:0> table_help

Help for table-reference commands.

HBase- CREATE TABLE & VERIFY

hbase(main):007:0> create 'emp','personal data','professional data'

hbase(main):008:0> describe 'emp'

To Verify table

hbase(main):008:0> list

TABLE

emp

hbase(main):014:0> exists 'emp'

Table emp does exist

HBase- STORE DATA IN TABLE

Insertion of data to first row

hbase(main):009:0> put 'emp','row1','personal data:name','raju'

hbase(main):010:0> put 'emp','row1','personal data:city','hyd'

hbase(main):011:0> put 'emp','row1','professional data:designation','manager'

hbase(main):012:0> put 'emp','row1','professional data:salary','50000'

hbase(main):013:0> scan 'emp'

ROW COLUMN+CELL

row1 column=personal data:city, timestamp=1523425226089, value=hyd

row1	column=personal data:name, timestamp=1523425200247, value=raju
row1	column=professional data:designation, timestamp=1523425255478, value=manager
row1	column=professional data:salary, timestamp=1523425270378, value=50000

Insertion of data to second row

```
hbase(main):013:0> put 'emp','row2','personal data:name','milind'  
hbase(main):018:0> put 'emp','row2','personal data:city','chennai'  
hbase(main):020:0> put 'emp','row2','professional data:designation','soft Engineer'  
hbase(main):015:0> put 'emp','row2','professional data:salary','30000'
```

Insertion of data to third row

```
hbase(main):022:0> put 'emp','row3','personal data:name','anita'  
hbase(main):023:0> put 'emp','row3','personal data:city','delhi'  
hbase(main):024:0> put 'emp','row3','professional data:designation','tester'  
hbase(main):025:0> put 'emp','row3','professional data:salary','40000'
```

TO READ THE DATA WHICH HAS BEEN ENTERED

➤ SCAN command reads all rows of that table

```
hbase(main):026:0> scan 'emp'  
ROW          COLUMN+CELL  
row1         column=personal data:city, timestamp=1523426980414, value=hyd  
row1         column=personal data:name, timestamp=1523426797711, value=raju  
row1         column=professional data:designation, timestamp=1523426835511, value=manager  
row1         column=professional data:salary, timestamp=1523426852045, value=50000  
row2         column=personal data:city, timestamp=1523427018284, value=chennai  
row2         column=personal data:name, timestamp=1523426910414, value=milind  
row2         column=professional data:designation, timestamp=1523427061723, value=soft Engineer  
row2         column=professional data:salary, timestamp=1523426965356, value=30000  
row3         column=personal data:city, timestamp=1523427129733, value=delhi
```

```
row3          column=personal data:name, timestamp=1523427115249, value=anita
row3          column=professional data:designation, timestamp=1523427162000, value=tester
row3          column=professional data:salary, timestamp=1523427187838, value=40000
3 row(s) in 0.0190 seconds
```

GET command is used to read data of a particular row or specific column

```
hbase(main):027:0> get 'emp','row3'
```

COLUMN	CELL
personal data:city	timestamp=1523427129733, value=delhi
personal data:name	timestamp=1523427115249, value=anita
professional data:designation	timestamp=1523427162000, value=tester
professional data:salary	timestamp=1523427187838, value=40000

```
hbase(main):004:0> get 'emp','row3',{COLUMN=>'personal data:city'}
```

COLUMN	CELL
personal data:city	timestamp=1523427129733, value=delhi

NOTE!!! COLUMN must be in CAPITAL letters

```
hbase(main):020:0> list
```

```
TABLE
emp
example
expert
```

TO UPDATE/MODIFY THE EXISTING DATA

Updating the row3 personal data column family's city column with different values

```
hbase(main):008:0> put 'emp','row3','personal data:city','2bang'
```

```
hbase(main):009:0> put 'emp','row3','personal data:city','3kochi'
hbase(main):010:0> get 'emp','row3',{COLUMN=>'personal data:city',VERSIONS=>3}
COLUMN                                CELL
personal data:city                    timestamp=1523428351922, value=3kochi
personal data:city                    timestamp=1523428337355, value=2bang
personal data:city                    timestamp=1523427129733, value=delhi
```

```
hbase(main):011:0> put 'emp','row3','personal data:city','4dehradun'
hbase(main):012:0> get 'emp','row3',{COLUMN=>'personal data:city',VERSIONS=>3}
COLUMN                                CELL
personal data:city                    timestamp=1523428391129, value=4dehradun
personal data:city                    timestamp=1523428351922, value=3kochi
personal data:city                    timestamp=1523428337355, value=2bang
```

```
hbase(main):013:0> get 'emp','row3',{COLUMN=>'personal data:city',VERSIONS=>2}
COLUMN                                CELL
personal data:city                    timestamp=1523428391129, value=4dehradun
personal data:city                    timestamp=1523428351922, value=3kochi
```

TO DISABLE THE TABLE

AFTER disable: "list" command will display the table name but "scan" command will show error

```
hbase(main):024:0> disable 'expert'
hbase(main):020:0> list
TABLE
emp
example
expert
```

```
hbase(main):027:0> scan 'expert'
ROW                                COLUMN+CELL
```


ERROR: org.apache.hadoop.hbase.DoNotRetryIOException: expert is disabled.

TO ALTER TABLE to add additional column family 'xyz with version 5'

hbase(main):017:0> disable 'emp'

hbase(main):018:0> alter 'emp',NAME=>'xyz',VERSIONS=>5

hbase(main):020:0> enable 'emp'

hbase(main):022:0> describe 'emp'

```
DESCRIPTION                                ENABLED
{NAME => 'personal data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFI
LTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0'
, TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', E
NCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'professional data', DATA_BLOCK_ENCODING =>
'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MI
N_VERSIONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMO
RY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'emp', FAMILIES => [{NAME => 'xyz',
DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NON true
E', REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '5', TTL => '2147483647', MIN_VERSI
ONS => '0', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', ENCODE_ON_DISK => 'true', IN_MEMORY
=> 'false', BLOCKCACHE => 'true'}],}}
```

hbase(main):023:0> put 'emp','row3','xyz:city','5mumbai'

hbase(main):024:0> put 'emp','row3','xyz:city','6jodhpur'

hbase(main):028:0> put 'emp','row3','xyz:city','7hubli'

hbase(main):029:0> put 'emp','row3','xyz:city','8london'

hbase(main):030:0> put 'emp','row3','xyz:city','9shimla'

hbase(main):031:0> put 'emp','row3','xyz:city','10kanpur'

hbase(main):032:0> put 'emp','row3','xyz:city','11nagpur'

hbase(main):033:0> get 'emp','row3',{COLUMN=>'xyz:city',VERSIONS=>5}

COLUMN

CELL

xyz:city

timestamp=1523431975773, value=11nagpur

xyz:city	timestamp=1523431968306, value=10kanpur
xyz:city	timestamp=1523431961690, value=9shimla
xyz:city	timestamp=1523431953548, value=8london
xyz:city	timestamp=1523431947056, value=7hubli

5 row(s) in 0.0080 seconds

hbase(main):034:0> scan 'emp'

ROW	COLUMN+CELL
row1	column=personal data:city, timestamp=1523426980414, value=hyd
row1	column=personal data:name, timestamp=1523426797711, value=raju
row1	column=professional data:designation, timestamp=1523426835511, value=manager
row1	column=professional data:salary, timestamp=1523426852045, value=50000
row2	column=personal data:city, timestamp=1523427018284, value=chennai
row2	column=personal data:name, timestamp=1523426910414, value=milind
row2	column=professional data:designation, timestamp=1523427061723, value=soft Engineer
row2	column=professional data:salary, timestamp=1523426965356, value=30000
row3	column=personal data:city, timestamp=1523428391129, value=4dehradun
row3	column=personal data:name, timestamp=1523427115249, value=anita
row3	column=professional data:designation, timestamp=1523427162000, value=tester
row3	column=professional data:salary, timestamp=1523427187838, value=40000
row3	column=xyz:city, timestamp=1523431975773, value=11nagpur

3 row(s) in 0.0620 seconds

TO DELETE

A PARTICULAR COLUMN FAMILY

hbase(main):037:0> disable 'emp'

hbase(main):038:0> alter 'emp','delete'=>'xyz'

hbase(main):039:0> enable 'emp'

hbase(main):042:0> get 'emp','row3'

COLUMN	CELL
personal data:city	timestamp=1523428391129, value=4dehradun
personal data:name	timestamp=1523427115249, value=anita

professional data:designation timestamp=1523427162000, value=tester
professional data:salary timestamp=1523427187838, value=40000
4 row(s) in 0.0080 seconds

A SPECIFIC CELL IN TABLE

hbase(main):045:0> delete 'emp','row3','personal data:name'

hbase(main):046:0> get 'emp','row3'

COLUMN	CELL
personal data:city	timestamp=1523428391129, value=4dehradun
professional data:designation	timestamp=1523427162000, value=tester
professional data:salary	timestamp=1523427187838, value=40000

3 row(s) in 0.1870 seconds

DELETE COMPLETE ROW IN TABLE

hbase(main):049:0> deleteall 'emp','row3'

hbase(main):050:0> get 'emp','row3'

COLUMN	CELL
--------	------

0 row(s) in 0.0340 seconds

hbase(main):051:0> scan 'emp'

ROW	COLUMN+CELL
row1	column=personal data:city, timestamp=1523426980414, value=hyd
row1	column=personal data:name, timestamp=1523426797711, value=raju
row1	column=professional data:designation, timestamp=1523426835511, value=manager
row1	column=professional data:salary, timestamp=1523426852045, value=50000
row2	column=personal data:city, timestamp=1523427018284, value=chennai
row2	column=personal data:name, timestamp=1523426910414, value=milind
row2	column=professional data:designation, timestamp=1523427061723, value=soft Engineer
row2	column=professional data:salary, timestamp=1523426965356, value=30000

2 row(s) in 0.0390 seconds

TO DROP THE TABLE

To drop , first table has to be disabled.

```
hbase(main):005:0> list
```

```
TABLE
```

```
emp
```

```
example
```

```
exp
```

```
expert
```

```
hbase(main):024:0> disable 'expert'
```

```
hbase(main):001:0> drop 'expert'
```

```
hbase(main):002:0> list
```

```
TABLE
```

```
emp
```

```
example
```

To drop all three table which starts with name 'ex'

```
hbase(main):006:0> disable_all 'ex.*'
```

```
example
```

```
exp
```

```
expert
```

Disable the above 3 tables (y/n)?

```
y
```

3 tables successfully disabled

```
hbase(main):007:0> drop_all 'ex.*'
```

```
example
```

```
exp
```

```
expert
```

Drop the above 3 tables (y/n)?

y

3 tables successfully dropped

hbase(main):052:0> count 'emp'

2 row(s) in 0.7080 seconds

