

Unit 5

Pig: Introduction to PIG, Execution Modes of Pig, Comparison of Pig with Databases, Grunt, Pig Latin, User Defined Functions, Data Processing operators.

HBase: Basics, Concepts, Clients, Example, HBase Versus RDBMS..

Introduction to PIG

Pig is a high-level data flow platform for executing Map Reduce programs of Hadoop. It was developed by Yahoo. The language for Pig is pig Latin.

Our Pig tutorial includes all topics of Apache Pig with Pig usage, Pig Installation, Pig Run Modes, Pig Latin concepts, Pig Data Types, Pig example, Pig user defined functions etc.

What is Apache Pig

Apache Pig is a high-level data flow platform for executing MapReduce programs of Hadoop. The language used for Pig is Pig Latin.



The Pig scripts get internally converted to Map Reduce jobs and get executed on data stored in HDFS. Apart from that, Pig can also execute its job in Apache Tez or Apache Spark.

Pig can handle any type of data, i.e., structured, semi-structured or unstructured and stores the corresponding results into Hadoop Data File System. Every task which can be achieved using PIG can also be achieved using java used in MapReduce.

Features of Apache Pig

Let's see the various uses of Pig technology.

1) Ease of programming

Writing complex java programs for map reduce is quite tough for non-programmers. Pig makes this process easy. In the Pig, the queries are converted to MapReduce internally.

2) Optimization opportunities

It is how tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.

3) Extensibility

A user-defined function is written in which the user can write their logic to execute over the data set.

4) Flexible

It can easily handle structured as well as unstructured data.

5) In-built operators

It contains various type of operators such as sort, filter and joins.

Differences between Apache MapReduce and PIG

Apache MapReduce	Apache PIG
It is a low-level data processing tool.	It is a high-level data flow tool.
Here, it is required to develop complex programs using Java or Python.	It is not required to develop complex programs.
It is difficult to perform data operations in MapReduce.	It provides built-in operators to perform data operations like union, sorting and ordering.

It doesn't allow nested data types.	It provides nested data types like tuple, bag, and map.
-------------------------------------	---

Advantages of Apache Pig

- Less code - The Pig consumes less line of code to perform any operation.
- Reusability - The Pig code is flexible enough to reuse again.
- Nested data types - The Pig provides a useful concept of nested data types like tuple, bag, and map.

Apache Pig Installation

In this section, we will perform the pig installation.

Pre-requisite

- **Java Installation** - Check whether the Java is installed or not using the following command.

1. `$java -version`

- **Hadoop Installation** - Check whether the Hadoop is installed or not using the following command.

1. `$hadoop version`

If any of them is not installed in your system, follow the below link to install it. [Click Here to Install](#)

Steps to install Apache Pig

- Download the [Apache Pig tar](#) file.
- Unzip the downloaded tar file.

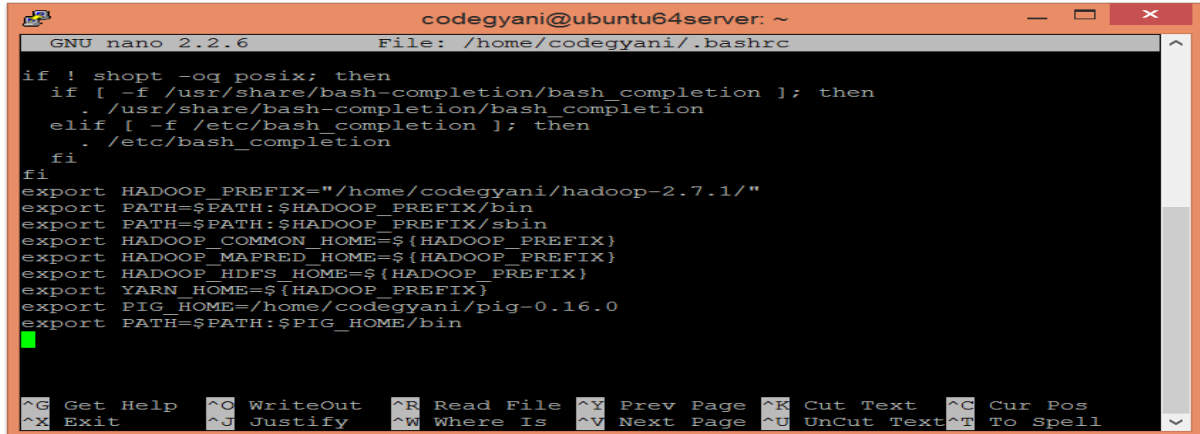
1. `$ tar -xvf pig-0.16.0.tar.gz`

- Open the bashrc file.

1. `$ sudo nano ~/.bashrc`

- Now, provide the following PIG_HOME path.

1. export **PIG_HOME**=/home/hduser/pig-0.16.0
2. export **PATH**=\$PATH:\$PIG_HOME/bin



```
codegyani@ubuntu64server: ~  
GNU nano 2.2.6 File: /home/codegyani/.bashrc  
if ! shopt -oq posix; then  
if [ -f /usr/share/bash-completion/bash_completion ]; then  
./usr/share/bash-completion/bash_completion  
elif [ -f /etc/bash_completion ]; then  
./etc/bash_completion  
fi  
fi  
export HADOOP_PREFIX="/home/codegyani/hadoop-2.7.1/"  
export PATH=$PATH:$HADOOP_PREFIX/bin  
export PATH=$PATH:$HADOOP_PREFIX/sbin  
export HADOOP_COMMON_HOME=${HADOOP_PREFIX}  
export HADOOP_MAPRED_HOME=${HADOOP_PREFIX}  
export HADOOP_HDFS_HOME=${HADOOP_PREFIX}  
export YARN_HOME=${HADOOP_PREFIX}  
export PIG_HOME=/home/codegyani/pig-0.16.0  
export PATH=$PATH:$PIG_HOME/bin  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Update the environment variable

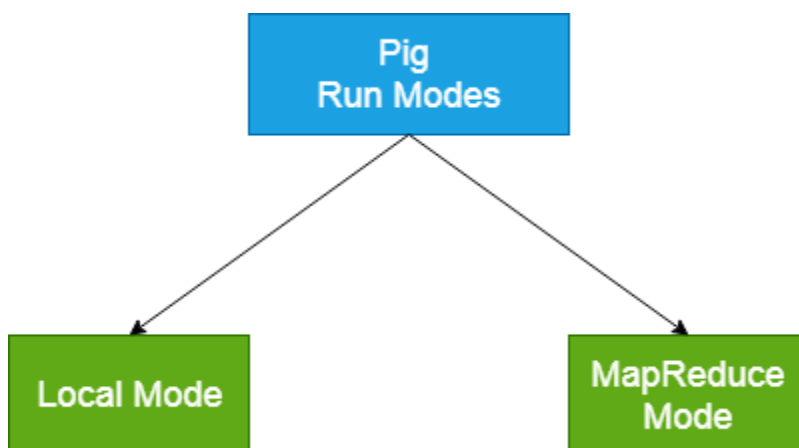
1. \$ source ~/.bashrc
 - Let's test the installation on the command prompt type

1. \$ pig -h
 - Let's start the pig in MapReduce mode.

1. \$ pig

Apache Pig Run Modes

Apache Pig executes in two modes: Local Mode and MapReduce Mode.



Local Mode

- It executes in a single JVM and is used for development experimenting and prototyping.
- Here, files are installed and run using localhost.
- The local mode works on a local file system. The input and output data stored in the local file system.

The command for local mode grunt shell:

1. `$ pig-x local`

MapReduce Mode

- The MapReduce mode is also known as Hadoop Mode.
- It is the default mode.
- In this Pig renders Pig Latin into MapReduce jobs and executes them on the cluster.
- It can be executed against semi-distributed or fully distributed Hadoop installation.
- Here, the input and output data are present on HDFS.

The command for Map reduce mode:

1. `$ pig`

Or,

1. `$ pig -x mapreduce`

Ways to execute Pig Program

These are the following ways of executing a Pig program on local and MapReduce mode: -

- **Interactive Mode** - In this mode, the Pig is executed in the Grunt shell. To invoke Grunt shell, run the pig command. Once the Grunt mode executes, we can provide Pig Latin statements and command interactively at the command line.

- **Batch Mode** - In this mode, we can run a script file having a .pig extension. These files contain Pig Latin commands.
- **Embedded Mode** - In this mode, we can define our own functions. These functions can be called as UDF (User Defined Functions). Here, we use programming languages like Java and Python.

Pig Latin

The Pig Latin is a data flow language used by Apache Pig to analyze the data in Hadoop. It is a textual language that abstracts the programming from the Java MapReduce idiom into a notation.

Pig Latin Statements

The Pig Latin statements are used to process the data. It is an operator that accepts a relation as an input and generates another relation as an output.

- It can span multiple lines.
- Each statement must end with a semi-colon.
- It may include expression and schemas.
- By default, these statements are processed using multi-query execution

Pig Latin Conventions

Convention	Description
()	The parenthesis can enclose one or more items. It can also be used to indicate the tuple data type. Example - (10, xyz, (3,6,9))
[]	The straight brackets can enclose one or more items. It can also be used to indicate the map data type. Example - [INNER OUTER]
{ }	The curly brackets enclose two or more items. It can also be used to indicate the bag data type. Example - { block nested_block }

...	<p>The horizontal ellipsis points indicate that you can repeat a portion of the code.</p> <p>Example - cat path [path ...]</p>
-----	--

Latin Data Types

Simple Data Types

Type	Description
int	<p>It defines the signed 32-bit integer.</p> <p>Example - 2</p>
long	<p>It defines the signed 64-bit integer.</p> <p>Example - 2L or 2l</p>
float	<p>It defines 32-bit floating point number.</p> <p>Example - 2.5F or 2.5f or 2.5e2f or 2.5.E2F</p>
double	<p>It defines 64-bit floating point number.</p> <p>Example - 2.5 or 2.5 or 2.5e2f or 2.5.E2F</p>
chararray	<p>It defines character array in Unicode UTF-8 format.</p> <p>Example - javatpoint</p>
bytearray	<p>It defines the byte array.</p>
boolean	<p>It defines the boolean type values.</p> <p>Example - true/false</p>
datetime	<p>It defines the values in datetime order.</p> <p>Example - 1970-01- 01T00:00:00.000+00:00</p>
biginteger	<p>It defines Java BigInteger values.</p> <p>Example - 5000000000000</p>

bigdecimal	It defines Java BigDecimal values. Example - 52.232344535345
------------	---

Complex Types

Type	Description
tuple	It defines an ordered set of fields. Example - (15,12)
bag	It defines a collection of tuples. Example - {(15,12), (12,15)}
map	It defines a set of key-value pairs. Example - [open#apache]

Pig Data Types

Apache Pig supports many data types. A list of Apache Pig Data Types with description and examples are given below.

Type	Description	Example
Int	Signed 32 bit integer	2
Long	Signed 64 bit integer	15L or 15l
Float	32 bit floating point	2.5f or 2.5F
Double	32 bit floating point	1.5 or 1.5e2 or 1.5E2
charArray	Character array	hello javatpoint
byteArray	BLOB(Byte array)	
tuple	Ordered set of fields	(12,43)
bag	Collection f tuples	{(12,43),(54,28)}

map	collection of tuples	[open#apache]
-----	----------------------	---------------

Pig Example

Use case: Using Pig find the most occurred start letter.

Solution:

Case 1: Load the data into bag named "lines". The entire line is stuck to element line of type character array.

1. `grunt> lines = LOAD "/user/Desktop/data.txt" AS (line: chararray);`

Case 2: The text in the bag lines needs to be tokenized this produces one word per row.

1. `grunt> tokens = FOREACH lines GENERATE flatten(TOKENIZE(line)) AS token: chararray;`

Case 3: To retain the first letter of each word type the below command .This commands uses substring method to take the first character.

1. `grunt> letters = FOREACH tokens GENERATE SUBSTRING(0,1) as letter : chararray;`

Case 4: Create a bag for unique character where the grouped bag will contain the same character for each occurrence of that character.

1. `grunt> lettergrp = GROUP letters by letter;`

Case 5: The number of occurrence is counted in each group.

1. `grunt> countletter = FOREACH lettergrp GENERATE group , COUNT(letters);`

Case 6: Arrange the output according to count in descending order using the commands below.

1. `grunt> OrderCnt = ORDER countletter BY $1 DESC;`

Case 7: Limit to One to give the result.

1. `grunt> result =LIMIT OrderCnt 1;`

Case 8: Store the result in HDFS . The result is saved in output directory under sonoo folder.

1. `grunt> STORE result into 'home/sonoo/output';`

Pig UDF (User Defined Functions)

To specify custom processing, Pig provides support for user-defined functions (UDFs). Thus, Pig allows us to create our own functions. Currently, Pig UDFs can be implemented using the following programming languages: -

- Java
- Python
- Jython
- JavaScript
- Ruby
- Groovy

Among all the languages, Pig provides the most extensive support for Java functions. However, limited support is provided to languages like Python, Jython, JavaScript, Ruby, and Groovy.

Example of Pig UDF

In Pig,

- All UDFs must extend "org.apache.pig.EvalFunc"
- All functions must override the "exec" method.

Let's see an example of a simple EVAL Function to convert the provided string to uppercase.

UPPER.java

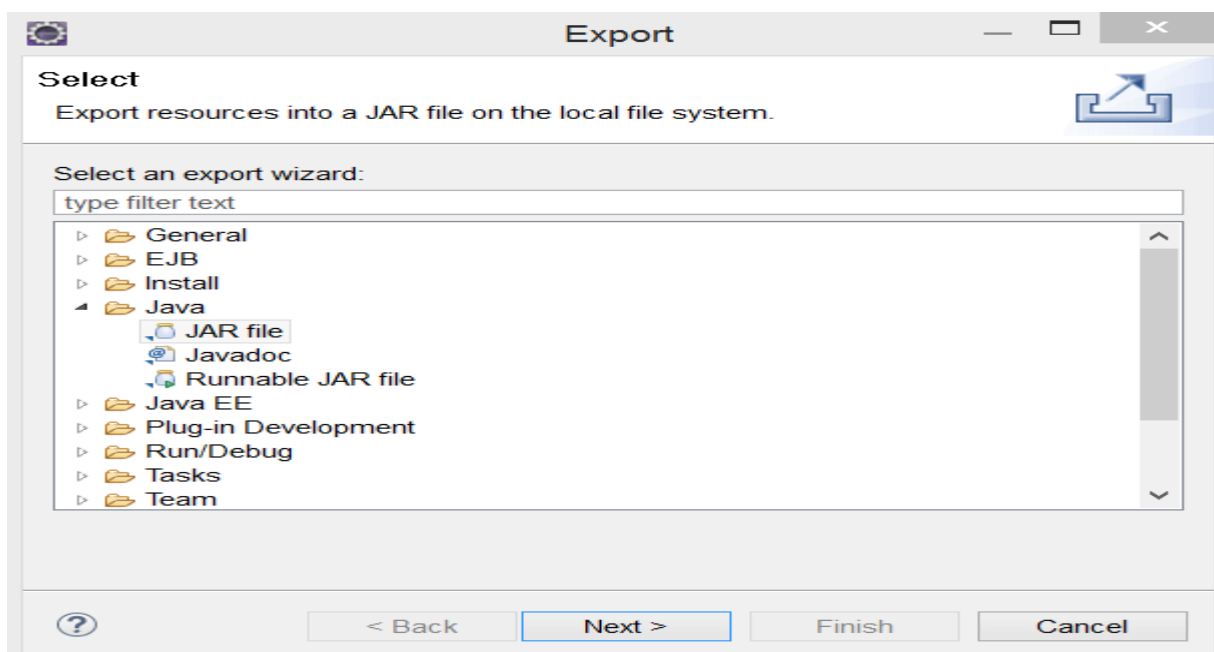
1. `package com.hadoop;`
- 2.
3. `import java.io.IOException;`
- 4.
5. `import org.apache.pig.EvalFunc;`
6. `import org.apache.pig.data.Tuple;`

```

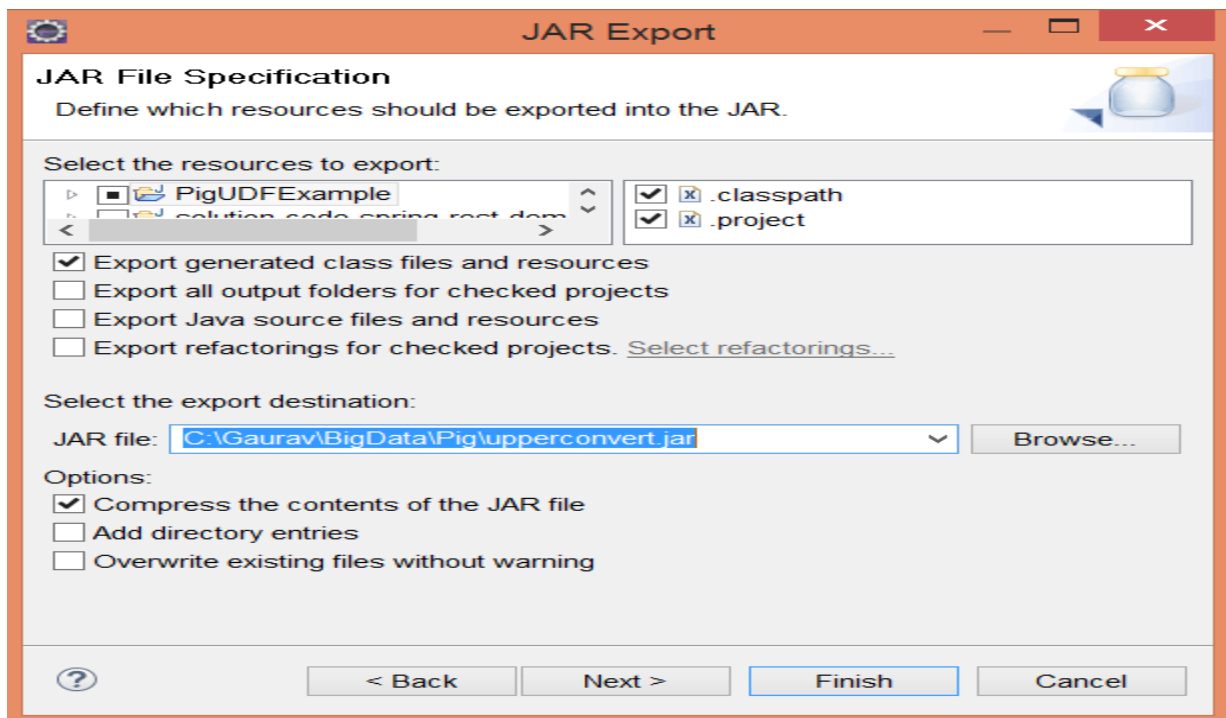
7.
8. public class TestUpper extends EvalFunc<String> {
9.     public String exec(Tuple input) throws IOException {
10.         if (input == null || input.size() == 0)
11.             return null;
12.         try{
13.             String str = (String)input.get(0);
14.             return str.toUpperCase();
15.         }catch(Exception e){
16.             throw new IOException("Caught exception processing input row ", e);
17.         }
18.     }
19. }

```

- Create the jar file and export it into the specific directory. For that ,right click on project - **Export - Java - JAR file - Next.**

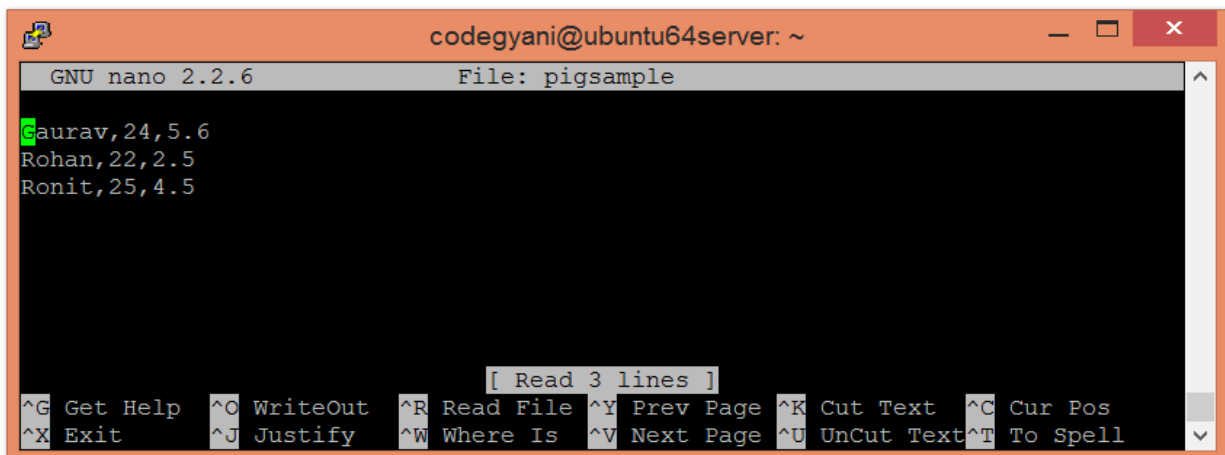


- Now, provide a specific name to the jar file and save it in a local system directory.



- Create a text file in your local machine and insert the list of tuples.

1. \$ nano pigsample



Apache Pig LOAD Operator

The Apache Pig LOAD operator is used to load the data from the file system.

Syntax

1. LOAD 'info' [USING FUNCTION] [AS SCHEMA];

Here,

- **LOAD** is a relational operator.

- **'info'** is a file that is required to load. It contains any type of data.
- **USING** is a keyword.
- **FUNCTION** is a load function.
- **AS** is a keyword.
- **SCHEMA** is a schema of passing file, enclosed in parentheses.

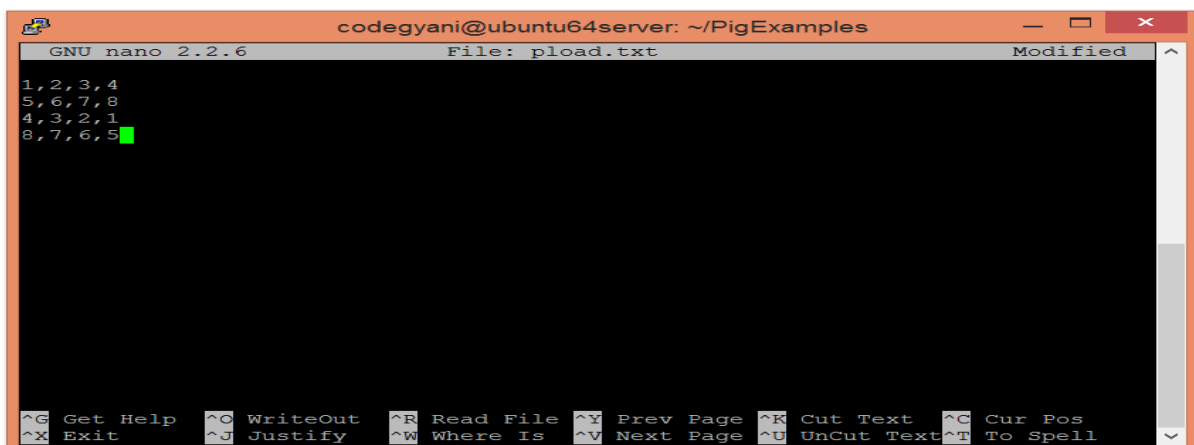
Example of LOAD Operator

In this example, we load the text file data from the file system.

Steps to execute LOAD Operator

- Create a text file in your local machine and provide some values to it.

1. \$ nano pload.txt



```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pload.txt Modified
1,2,3,4
5,6,7,8
4,3,2,1
8,7,6,5
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Upload the text files on HDFS in the specific directory.

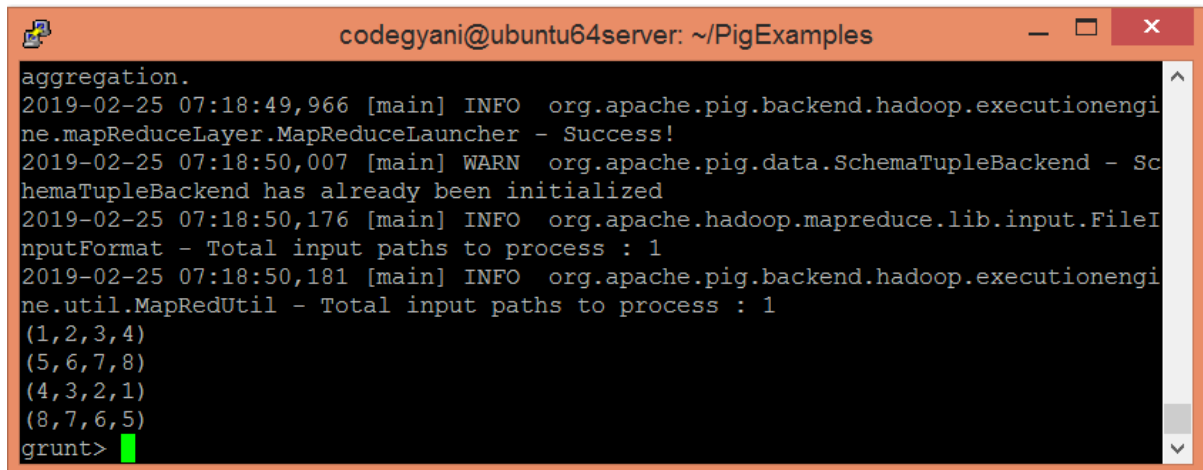
1. \$ hdfs dfs -put pload.txt /pigexample

- Open the pig MapReduce run mode.

1. \$ pig

- Load the file that contains the data.

1. grunt> **A** = **LOAD** '/pigexample/pload.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int,a4:int);
 - Now, execute and verify the data.
1. grunt> **DUMP A**;



```
codegyani@ubuntu64server: ~/PigExamples
aggregation.
2019-02-25 07:18:49,966 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 07:18:50,007 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2019-02-25 07:18:50,176 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 07:18:50,181 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(1,2,3,4)
(5,6,7,8)
(4,3,2,1)
(8,7,6,5)
grunt>
```

Apache Pig CROSS Operator

The Apache Pig CROSS operator facilitates to compute the cross product of two or more relations. Using CROSS operator is an expensive operation and should be used sparingly.

Example of CROSS Operator

In this example, we compute the data of two relations.

Steps to execute CROSS Operator

- Create a text file in your local machine and write some values into it.
1. \$ nano pcross1.txt



```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pcross1.txt
2,5
3,6
[ Read 2 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

1. \$ nano pcross2.txt



```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pcross2.txt
3,6,8
2,6,9
[ Read 2 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values written in both text files.

1. \$ cat pcross1.txt

2. \$ cat pcross2.txt

A terminal window titled 'codegyani@ubuntu64server: ~/PigExamples' with standard window controls. The terminal shows the following commands and output:

```
codegyani@ubuntu64server:~/PigExamples$ cat pcross1.txt
2,5
3,6
codegyani@ubuntu64server:~/PigExamples$ cat pcross2.txt
3,6,8
2,6,9
codegyani@ubuntu64server:~/PigExamples$
```

- Upload both text files on HDFS in the specific directory.

1. \$ hdfs dfs -put pcross1.txt /pigexample
2. \$ hdfs dfs -put pcross2.txt /pigexample

- Open the pig MapReduce run mode.

1. \$ pig

- Load the file that contains the data.

1. grunt> **A** = **LOAD** '/pigexample/pcross1.txt' USING PigStorage(',') AS (a1:int,a2:int);

- Now, execute and verify the data.

1. grunt> **DUMP** A;

- Load the another file that contains the data.

1. grunt> **B** = **LOAD** '/pigexample/pcross2.txt' USING PigStorage(',') AS (b1:int,b2:int,b3:int);

- Now, execute and verify the data.

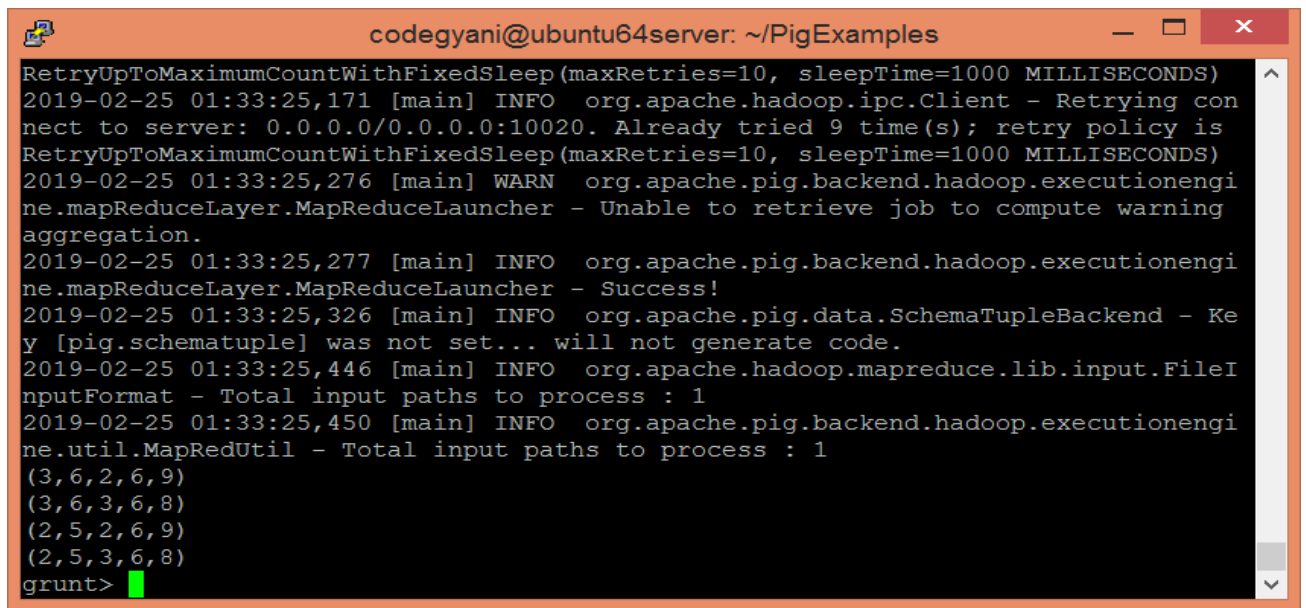
1. grunt> **DUMP** B;

- Let's perform the Cartesian product between both files.

1. grunt> **Result** = **CROSS** A,B;

- Now, execute and verify the data.

1. grunt> DUMP Result;

A terminal window titled 'codegyani@ubuntu64server: ~/PigExamples' displays a series of log messages. The logs include retry attempts for a server connection, a warning about job retrieval, and several informational messages about schema tuples and input paths. The output concludes with a list of five coordinate tuples: (3,6,2,6,9), (3,6,3,6,8), (2,5,2,6,9), (2,5,3,6,8), and (2,5,3,6,8). The prompt 'grunt>' is followed by a green cursor.

```
codegyani@ubuntu64server: ~/PigExamples
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 01:33:25,171 [main] INFO org.apache.hadoop.ipc.Client - Retrying connect to server: 0.0.0.0/0.0.0.0:10020. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 01:33:25,276 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-25 01:33:25,277 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 01:33:25,326 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 01:33:25,446 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 01:33:25,450 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(3,6,2,6,9)
(3,6,3,6,8)
(2,5,2,6,9)
(2,5,3,6,8)
grunt>
```

Here, we got the desired output.



HIVE vs PIG vs SQL

Hive, Pig, and SQL are the popular technologies for querying and programming to tame Big Data. Let's have a comparison between three Hive vs Pig vs SQL and choose the best for Big Data project.

CRITERIA	HIVE	PIG	SQL
Languages used	Uses HiveQL, a declarative language	Uses Pig latin, a procedural data flow languages	SQL itself is a declarative language
Definition	An open source built with an analytical focus used for Analytical queries	An open source and high-level data flow language with a Multi-query approach	General purpose database language for analytical and transactional queries
Developed by	Facebook	Yahoo	Oracle
Suitable for	Batch processing OLAP (Online Analytical Processing)	Complex & nested data structure	Business demands for fast data analysis
Operational for	Structured data	Structured and semi-structured data	Relational database management
Compatibility with MapReduce	Yes	Yes	Yes
Schema Support	Support schema for data insertion	Doesn't support schema	Strictly support schema for data storage
Mainly Used by	Data Analysts	Researchers and Programmers	Data Analysts, Data Scientists, and Programmers

What is HBase

Hbase is an open source and sorted map data built on Hadoop. It is column oriented and horizontally scalable.

It is based on Google's Big Table. It has set of tables which keep data in key value format. Hbase is well suited for sparse data sets which are very common in big data use cases. Hbase provides APIs enabling development in practically any programming language. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

Why HBase

- RDBMS get exponentially slow as the data becomes large
- Expects data to be highly structured, i.e. ability to fit in a well-defined schema
- Any change in schema might require a downtime
- For sparse datasets, too much of overhead of maintaining NULL values

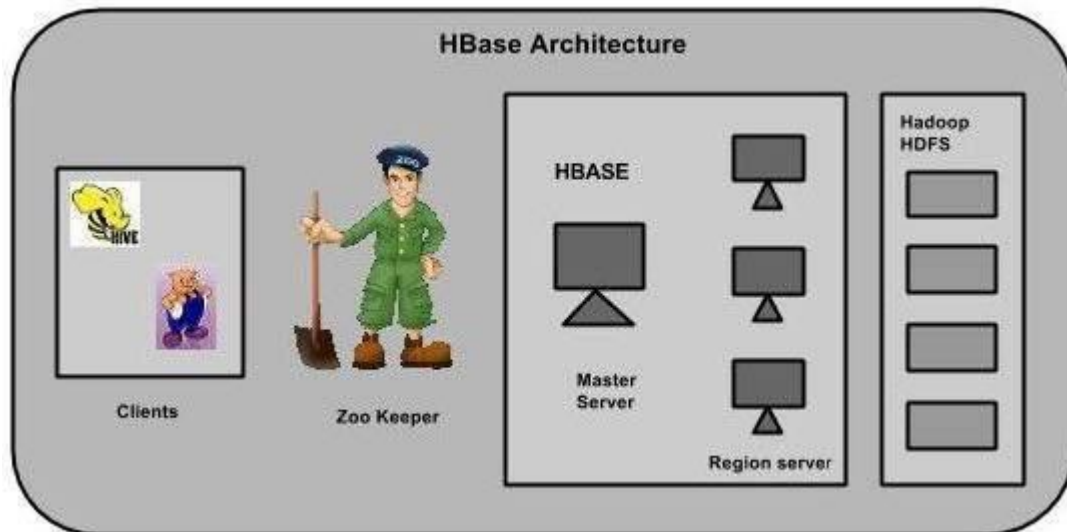
Features of Hbase

- Horizontally scalable: You can add any number of columns anytime.
- Automatic Failover: Automatic failover is a resource that allows a system administrator to automatically switch data handling to a standby system in the event of system compromise
- Integrations with Map/Reduce framework: All the commands and java codes internally implement Map/ Reduce to do the task and it is built over Hadoop Distributed File System.
- sparse, distributed, persistent, multidimensional sorted map, which is indexed by rowkey, column key, and timestamp.
- Often referred as a key value store or column family-oriented database, or storing versioned maps of maps.
- fundamentally, it's a platform for storing and retrieving data with random access.
- It doesn't care about datatypes (storing an integer in one row and a string in another for the same column).

- It doesn't enforce relationships within your data.
- It is designed to run on a cluster of computers, built using commodity hardware.

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.

Note: The term ‘store’ is used for regions to explain the storage structure.



HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

MasterServer

The master server -

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

Regions are nothing but tables that are split up and spread across the region servers.

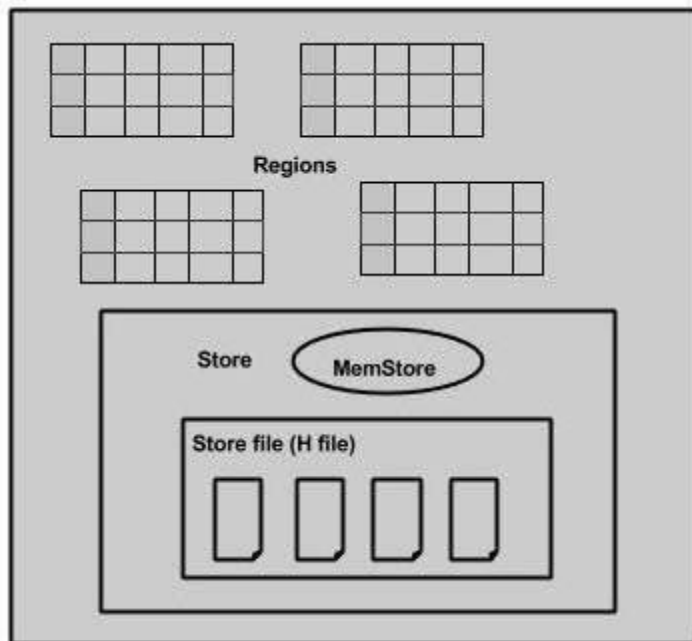
Region server

The region servers have regions that -

- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.

- Decide the size of the region by following the region size thresholds.

When we take a deeper look into the region server, it contains regions and stores as shown below:

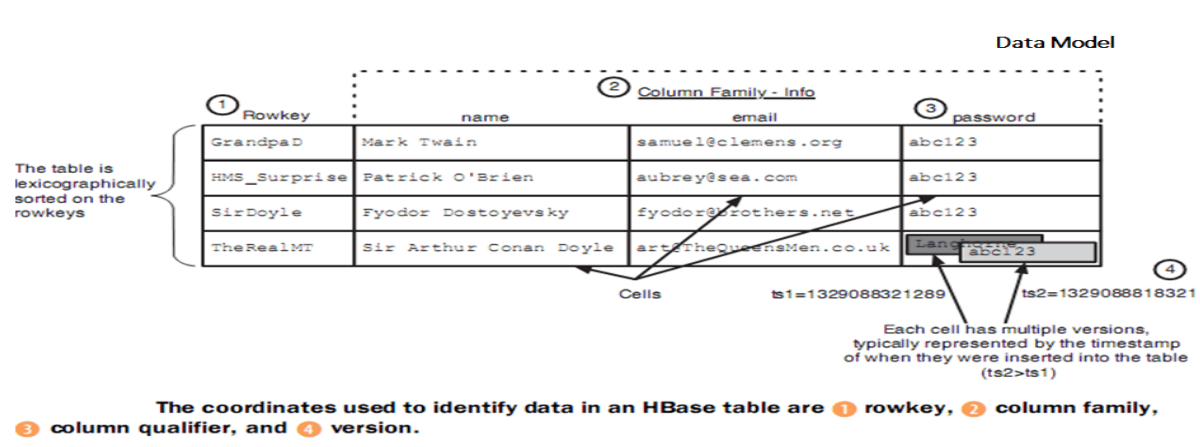


The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

Zookeeper

- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

HBase Data Model



HBase - Client API

This chapter describes the java client API for HBase that is used to perform **CRUD** operations on HBase tables. HBase is written in Java and has a Java Native API. Therefore it provides programmatic access to Data Manipulation Language (DML).

Class HBase Configuration

Adds HBase configuration files to a Configuration. This class belongs to the **org.apache.hadoop.hbase** package.

Methods and description

S.No.	Methods and Description
1	static org.apache.hadoop.conf.Configuration create() This method creates a Configuration with HBase resources.

Class HTable

HTable is an HBase internal class that represents an HBase table. It is an implementation of table that is used to communicate with a single HBase table. This class belongs to the **org.apache.hadoop.hbase.client** class.

Constructors

S.No.	Constructors and Description
-------	------------------------------

1	HTable()
2	HTable(TableName tableName, ClusterConnection connection, ExecutorService pool) Using this constructor, you can create an object to access an HBase table.

Methods and description

S.No.	Methods and Description
1	void close() Releases all the resources of the HTable.
2	void delete(Delete delete) Deletes the specified cells/row.
3	boolean exists(Get get) Using this method, you can test the existence of columns in the table, as specified by Get.
4	Result get(Get get) Retrieves certain cells from a given row.
5	org.apache.hadoop.conf.Configuration getConfiguration() Returns the Configuration object used by this instance.
6	TableName getName() Returns the table name instance of this table.
7	HTableDescriptor getTableDescriptor() Returns the table descriptor for this table.
8	byte[] getTableName() Returns the name of this table.

9	void put(Put put) Using this method, you can insert data into the table.
---	--

Class Put

This class is used to perform Put operations for a single row. It belongs to the **org.apache.hadoop.hbase.client** package.

Constructors

S.No.	Constructors and Description
1	Put(byte[] row) Using this constructor, you can create a Put operation for the specified row.
2	Put(byte[] rowArray, int rowOffset, int rowLength) Using this constructor, you can make a copy of the passed-in row key to keep local.
3	Put(byte[] rowArray, int rowOffset, int rowLength, long ts) Using this constructor, you can make a copy of the passed-in row key to keep local.
4	Put(byte[] row, long ts) Using this constructor, we can create a Put operation for the specified row, using a given timestamp.

Methods

S.No.	Methods and Description
1	Put add(byte[] family, byte[] qualifier, byte[] value) Adds the specified column and value to this Put operation.
2	Put add(byte[] family, byte[] qualifier, long ts, byte[] value) Adds the specified column and value, with the specified timestamp as its version to this Put operation.

3	Put add(byte[] family, ByteBuffer qualifier, long ts, ByteBuffer value) Adds the specified column and value, with the specified timestamp as its version to this Put operation.
4	Put add(byte[] family, ByteBuffer qualifier, long ts, ByteBuffer value) Adds the specified column and value, with the specified timestamp as its version to this Put operation.

Class Get

This class is used to perform Get operations on a single row. This class belongs to the **org.apache.hadoop.hbase.client** package.

Constructor

S.No.	Constructor and Description
1	Get(byte[] row) Using this constructor, you can create a Get operation for the specified row.
2	Get(Get get)

Methods

S.No.	Methods and Description
1	Get addColumn(byte[] family, byte[] qualifier) Retrieves the column from the specific family with the specified qualifier.
2	Get addFamily(byte[] family) Retrieves all columns from the specified family.

Class Delete

This class is used to perform Delete operations on a single row. To delete an entire row, instantiate a Delete object with the row to delete. This class belongs to the **org.apache.hadoop.hbase.client** package.

Constructor

S.No.	Constructor and Description
1	Delete(byte[] row) Creates a Delete operation for the specified row.
2	Delete(byte[] rowArray, int rowOffset, int rowLength) Creates a Delete operation for the specified row and timestamp.
3	Delete(byte[] rowArray, int rowOffset, int rowLength, long ts) Creates a Delete operation for the specified row and timestamp.
4	Delete(byte[] row, long timestamp) Creates a Delete operation for the specified row and timestamp.

Methods

S.No.	Methods and Description
1	Delete addColumn(byte[] family, byte[] qualifier) Deletes the latest version of the specified column.
2	Delete addColumns(byte[] family, byte[] qualifier, long timestamp) Deletes all versions of the specified column with a timestamp less than or equal to the specified timestamp.
3	Delete addFamily(byte[] family) Deletes all versions of all columns of the specified family.
4	Delete addFamily(byte[] family, long timestamp) Deletes all columns of the specified family with a timestamp less than or equal to the specified timestamp.

Class Result

This class is used to get a single row result of a Get or a Scan query.

Constructors

S.No.	Constructors
1	Result() Using this constructor, you can create an empty Result with no KeyValue payload; returns null if you call raw Cells().

Methods

S.No.	Methods and Description
1	byte[] getValue(byte[] family, byte[] qualifier) This method is used to get the latest version of the specified column.
2	byte[] getRow() This method is used to retrieve the row key that corresponds to the row from which this Result was created.

HBase Example

Let's see a HBase example to import data of a file in HBase table.

Use Case

We have to import data present in the file into an HBase table by creating it through Java API.

Data_file.txt contains the below data

- 1,India,Bihar,Champaran,2009,April,P1,1,5
- 2,India, Bihar,Patna,2009,May,P1,2,10
- 3,India, Bihar,Bhagalpur,2010,June,P2,3,15
- 4,United States,California,Fresno,2009,April,P2,2,5
- 5,United States,California,Long Beach,2010,July,P2,4,10
- 6,United States,California,San Francisco,2011,August,P1,6,20

The Java code is shown belowThis data has to be inputted into a new HBase table to be created through JAVA API. Following column families have to be created

1. "sample,region,time.product,sale,profit".

Column family region has three column qualifiers: country, state, city

Column family Time has two column qualifiers: year, month

Jar Files

Make sure that the following jars are present while writing the code as they are required by the HBase.

- a. commons-logging-1.0.4
- b. commons-logging-api-1.0.4
- c. hadoop-core-0.20.2-cdh3u2
- d. hbase-0.90.4-cdh3u2
- e. log4j-1.2.15
- f. zookeeper-3.3.3-cdh3u0

Program Code

1. import java.io.BufferedReader;
2. import java.io.File;
3. import java.io.FileReader;
4. import java.io.IOException;
5. import java.util.StringTokenizer;
- 6.
7. import org.apache.hadoop.conf.Configuration;
8. import org.apache.hadoop.hbase.HBaseConfiguration;
9. import org.apache.hadoop.hbase.HColumnDescriptor;
10. import org.apache.hadoop.hbase.HTableDescriptor;
11. import org.apache.hadoop.hbase.client.HBaseAdmin;
12. import org.apache.hadoop.hbase.client.HTable;
13. import org.apache.hadoop.hbase.client.Put;
14. import org.apache.hadoop.hbase.util.Bytes;
- 15.
- 16.
17. public class readFromFile {
18. public static void main(String[] args) throws IOException{

```

19.     if(args.length==1)
20.     {
21.         Configuration conf = HBaseConfiguration.create(new Configuration());
22.         HBaseAdmin hba = new HBaseAdmin(conf);
23.         if(!hba.tableExists(args[0])){
24.             HTableDescriptor ht = new HTableDescriptor(args[0]);
25.             ht.addFamily(new HColumnDescriptor("sample"));
26.             ht.addFamily(new HColumnDescriptor("region"));
27.             ht.addFamily(new HColumnDescriptor("time"));
28.             ht.addFamily(new HColumnDescriptor("product"));
29.             ht.addFamily(new HColumnDescriptor("sale"));
30.             ht.addFamily(new HColumnDescriptor("profit"));
31.             hba.createTable(ht);
32.             System.out.println("New Table Created");
33.
34.             HTable table = new HTable(conf,args[0]);
35.
36.             File f = new File("/home/training/Desktop/data");
37.             BufferedReader br = new BufferedReader(new FileReader(f));
38.             String line = br.readLine();
39.             int i = 1;
40.             String rowname="row";
41.             while(line!=null && line.length()!=0){
42.                 System.out.println("Ok till here");
43.                 StringTokenizer tokens = new StringTokenizer(line,"");
44.                 rowname = "row"+i;
45.                 Put p = new Put(Bytes.toBytes(rowname));
46.                 p.add(Bytes.toBytes("sample"),Bytes.toBytes("sampleNo."),
47. Bytes.toBytes(Integer.parseInt(tokens.nextToken())));
48.                 p.add(Bytes.toBytes("region"),Bytes.toBytes("country"),Bytes.toBytes(tokens.nextToken()));
49.                 p.add(Bytes.toBytes("region"),Bytes.toBytes("state"),Bytes.toBytes(tokens.nextToken()));
50.                 p.add(Bytes.toBytes("region"),Bytes.toBytes("city"),Bytes.toBytes(tokens.nextTok
extToken()));
51.                 p.add(Bytes.toBytes("time"),Bytes.toBytes("year"),Bytes.toBytes(Integer.pa
rseInt(tokens.nextToken())));

```

```

52.         p.add(Bytes.toBytes("time"),Bytes.toBytes("month"),Bytes.toBytes(tokens.
nextToken()));
53.         p.add(Bytes.toBytes("product"),Bytes.toBytes("productNo."),Bytes.toBytes
(tokens.nextToken()));
54.         p.add(Bytes.toBytes("sale"),Bytes.toBytes("quantity"),Bytes.toBytes(Intege
r.parseInt(tokens.nextToken())));
55.         p.add(Bytes.toBytes("profit"),Bytes.toBytes("earnings"),Bytes.toBytes(toke
ns.nextToken()));
56.         i++;
57.         table.put(p);
58.         line = br.readLine();
59.     }
60.     br.close();
61.     table.close();
62. }
63. else
64.     System.out.println("Table Already exists.Please enter another table name");

65. }
66. else
67.     System.out.println("Please Enter the table name through command line");
68. }
69. }

```

HBase Versus RDBMS..

Sr. No.	Key	RDBMS	HBase
1	Definition	RDBMS stands for Relational DataBase Management System.	HBase has no full form.
2	SQL	RDBMS requires SQL, Structured Query Language.	HBase does not need SQL.
3	Schema	RDBMS has a fixed schema.	HBase has no fixed schema.
4	Orientation	RDBMS is row oriented.	HBase is column oriented.
5	Scalability	RDBMS faces problems in scalability.	HBase is highly scalable.

Sr. No.	Key	RDBMS	HBase
6	Nature	DBMS is static in nature.	HBase is dynamic in nature.
7	Data Retrieval	RDBMS data retrieval is slow.	HBase data retrieval is fast.
8	RULE	RDBMS follows ACID(Atomicity, Consistency, Isolation and Durability) Rule.	HBase follows CAP(Consistency, Availability, Partition-tolerance) Rule.
9	Data structure	RDBMS handles structural data.	HBase handles structural, non-structural and semi-structural data.
10	Sparse Data Handling	Sparse data handling is not present.	Sparse data handling is present.

- Schema/Database in RDBMS can be compared to namespace in Hbase.
- A table in RDBMS can be compared to column family in Hbase.
- A record (after table joins) in RDBMS can be compared to a record in Hbase.
- A collection of tables in RDBMS can be compared to a table in Hbase..

