**UNIT - V**

**What is HBase?**

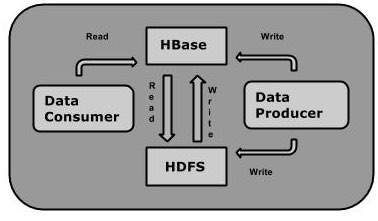
HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data. This tutorial provides an introduction to HBase, the procedures to set up HBase on Hadoop File Systems, and ways to interact with HBase shell. It also describes how to connect to HBase using java, and how to perform basic operations on HBase using java. HBase is an **open-source**, **distributed**, **column-oriented** database built on top of HDFS based on BigTable!

HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

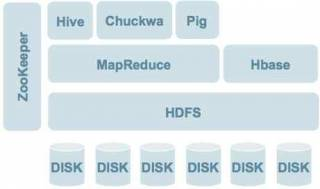
One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



**HBase and HDFS**

|  |  |
| --- | --- |
| **HDFS** | **HBase** |
| HDFS is a distributed file system suitable for storing large files. | HBase is a database built on top of the HDFS. |
| HDFS does not support fast individual record lookups. | HBase provides fast lookups for larger tables. |
| It provides high latency batch processing; no concept of batch processing. | It provides low latency access to single rows from billions of records (Random access). |
| It provides only sequential access of data. | HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups. |

**HBASE Framework**



## HBase Data Model

## Data is divided into various *tables*

## Table is composed of *columns,* columns are grouped into *column-families*

## 

## Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an 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.

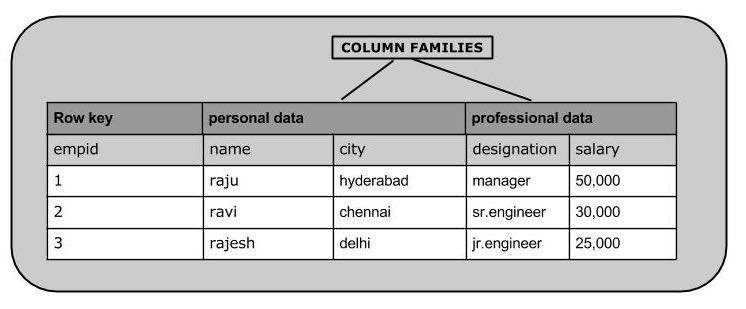
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rowid** | **Column Family** | | | **Column Family** | | | **Column Family** | | | **Column Family** | | |
| **col1** | **col2** | **col3** | **col1** | **col2** | **col3** | **col1** | **col2** | **col3** | **col1** | **col2** | **col3** |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |

## Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

|  |  |
| --- | --- |
| **Row-Oriented Database** | **Column-Oriented Database** |
| It is suitable for Online Transaction Process (OLTP). | It is suitable for Online Analytical Processing (OLAP). |
| Such databases are designed for small number of rows and columns. | Column-oriented databases are designed for huge tables. |

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



**HBase and RDBMS**

|  |  |
| --- | --- |
| **Hbase** | **RDBMS** |
| HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families. | An RDBMS is governed by its schema, which describes the whole structure of tables. |
| It is built for wide tables. HBase is horizontally scalable. | It is thin and built for small tables. Hard to scale. |
| No transactions are there in HBase. | RDBMS is transactional. |
| It has de-normalized data. | It will have normalized data. |
| It is good for semi-structured as well as structured data. | It is good for structured data. |

**Features of HBase**

* HBase is linearly scalable.
* It has automatic failure support.
* It provides consistent read and writes.
* It integrates with Hadoop, both as a source and a destination.
* It has easy java API for client.
* It provides data replication across clusters.

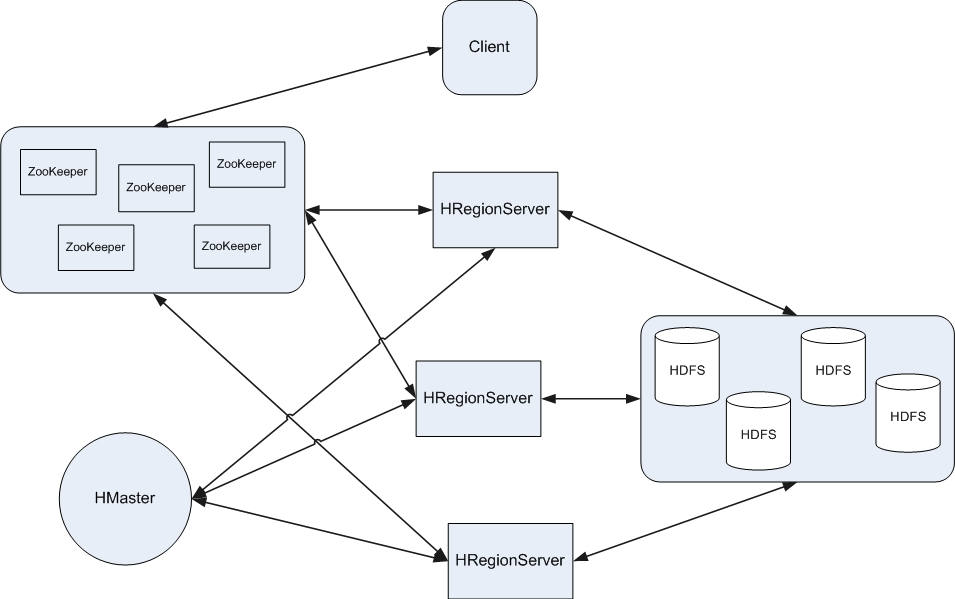
**Where to Use HBase**

* Apache HBase is used to have random, real-time read/write access to Big Data.
* It hosts very large tables on top of clusters of commodity hardware.
* Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

**Applications of HBase**

* It is used whenever there is a need to write heavy applications.
* HBase is used whenever we need to provide fast random access to available data.
* Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

**HBase Architecture**

****

* **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.**

**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 contain regions and stores
* 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.

**CASSANDRA**

The data model of Cassandra is significantly different from what we normally see in an RDBMS. This chapter provides an overview of how Cassandra stores its data.

## Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

### Keyspace

Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are −

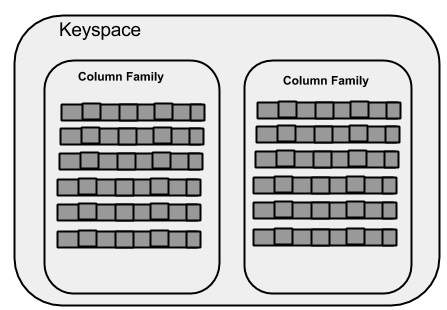
* **Replication factor** − It is the number of machines in the cluster that will receive copies of the same data.
* **Replica placement strategy** − It is nothing but the strategy to place replicas in the ring. We have strategies such as **simple strategy** (rack-aware strategy), **old network topology strategy** (rack-aware strategy), and **network topology strategy** (datacenter-shared strategy).
* **Column families** − Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows −

CREATE KEYSPACE Keyspace name

WITH replication = {'class': 'SimpleStrategy', 'replication\_factor' : 3};

The following illustration shows a schematic view of a Keyspace.



### Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

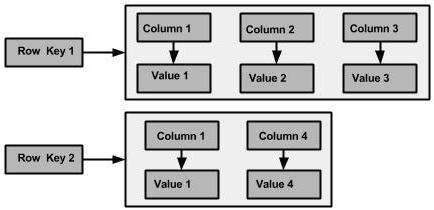
|  |  |
| --- | --- |
| **Relational Table** | **Cassandra column Family** |
| A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value. | In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time. |
| Relational tables define only columns and the user fills in the table with values. | In Cassandra, a table contains columns, or can be defined as a super column family. |

A Cassandra column family has the following attributes −

* **keys\_cached** − It represents the number of locations to keep cached per SSTable.
* **rows\_cached** − It represents the number of rows whose entire contents will be cached in memory.
* **preload\_row\_cache** − It specifies whether you want to pre-populate the row cache.

**Note −**Unlike relational tables where a column family’s schema is not fixed, Cassandra does not force individual rows to have all the columns.

The following figure shows an example of a Cassandra column family.



### Column

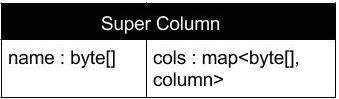
A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.



### SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns.

Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same column family, and a super column can be helpful here.Given below is the structure of a super column.



## Data Models of Cassandra and RDBMS

The following table lists down the points that differentiate the data model of Cassandra from that of an RDBMS.

|  |  |
| --- | --- |
| **RDBMS** | **Cassandra** |
| RDBMS deals with structured data. | Cassandra deals with unstructured data. |
| It has a fixed schema. | Cassandra has a flexible schema. |
| In RDBMS, a table is an array of arrays. (ROW x COLUMN) | In Cassandra, a table is a list of “nested key-value pairs”. (ROW x COLUMN key x COLUMN value) |
| Database is the outermost container that contains data corresponding to an application. | Keyspace is the outermost container that contains data corresponding to an application. |
| Tables are the entities of a database. | Tables or column families are the entity of a keyspace. |
| Row is an individual record in RDBMS. | Row is a unit of replication in Cassandra. |
| Column represents the attributes of a relation. | Column is a unit of storage in Cassandra. |
| RDBMS supports the concepts of foreign keys, joins. | Relationships are represented using collections. |

**HIVE**

HIVE is a data warehousing tool used to query structured data built on top of Hadoop. It processes batch jobs on huge data.

* Facebook created Hive component to manage their ever-growing volumes of log data.Hive makes use of the following:

1. HDFS for storage
2. MapReduce for execution
3. Stores metadata in an RDBMS

2007-Hive was born at FB to analyze their incoming log data

2008- Hive became Apache Hadoop sub-project

**HIVE FEATURES**

* Hive provides HQL (Hive Query Language) which is similar to SQL.
* Hive compiles SQL queries into MapReduce jobs and then runs the job in the Hadoop cluster. HQL is easy to code
* Hive provides extensive data type functions and formats for data summarization and analysis.Hive supports rich data types such as structs, lists and maps
* Hive supports SQL filters, group-by and order-by clauses. Custom types, Custom functions can be defined.

**HIVE Integration and Work Flow**

* Hourly Log Data can be stored directly into HDFS and then data cleansing is performed on the log file. Finally Hive table can be created to query the log file.

**HIVE DATA UNITS**

* Databases- The namespace for tables
* Tables- set of records that have similar schema
* Partitions- Logical separations of data based on classification of given information as per specific attributes .Once hive has partitioned the data based on a specified key, it starts to assemble the records into specific folders as and when the records are inserted.
* Buckets(Clusters) – similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.
* A Database contains several tables.Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as sub directory, Bucketed tables are stored as a File.

**HIVE ARCHITECTURE**

* Hive CLI (Command Line interface)-The most commonly used interface to interact with Hive
* Hive Web Interface- Simple graphic user interface to interact with Hive and to execute query
* Hive Server- This is an optional server.This can be used to submit Hive jobs from a remote client
* JDBC/ODBC – Jobs can be submitted from a JDBC Client.One can write a java code to connect to Hive and submit jobs on it.
* Driver- Hive queries are sent to the driver for compilation, optimization and execution
* Metastore – Hive table definitions and mappings to the data are stored in a metastore.A Metastore consists of the following:
* - Metastore service: offers interface to the Hive
* - Database – Stores data definitions, mappings to the data and others
* The meta store is updated whenever a table is created or deleted from Hive.



**Three Kinds of Metastores**

* Embedded Metastore- mainly used for unit tests.Default metastore .Only one process is allowed to connect to the metastore at a time
* Local Metastore- Metadata can be stored in any RDBMS component like MySQL.Local metastore allows multiple connections at a time.
* Remote Metastore- The Hive driver and metastore interface run on different JVMs .

**Hive Data Types**

* Primitive Data Types :

Numeric Data Type

- TINYINT- 1 BYTE SIGNED INTEGER

- SMALLINT – 2 –Byte Signed integer

- INT – 4 Byte signed integer

- BIGINT – 8 Byte signed integer

- FLOAT – 4 Byte single precision floating point

- DOUBLE – 8 Byte double precision floating point

* String Types

- STRING

- VARCHAR

- CHAR

* Miscellaneous Types

- BOOLEAN

- BINARY

Collection Data Types :

* STRUCT – Similar to C Struct
* MAP – A collection of key-value pairs
* ARRAY – Ordered sequence of same types

**Hive File Format**

* TEXT FILE – Default .In this format , each record is a line in the file .In Text file , different control characters are used as delimiters.
* SEQUENTIAL FILE – Sequential files are flat files that store binary key-value pairs. It includes compression support which reduces the CPU , I/O requirement.
* RCFile (Record Columnar File)- Stores the data in Column oriented manner which ensures that Aggregation operation is not an expensive operation.

In RCFile, instead of only partitioning the table horizontally like the row oriented DBMS (row-store) , RCFile partitions this table first horizontally and then vertically to serialize the data.Based on the user specified value, first the table is partitioned into multiple row groups horizontally.Next , in every row group RCFile partitions the data vertically like column-store.

**HIVE QUERY LANGAUGE**

* HQL provides basic SQL like operations :

1. Create and manage tables and partitions
2. Support various Relational, Arithmetic and Logical operators
3. Evaluate Functions
4. Download the contents of a table to a local directory or result of queries to HDFS directory

DDL Statements

* Create/Drop/Alter Database
* Create/Drop/Truncate Table
* Alter Table/Partition/Column
* Create/Drop/Alter view
* Create/ Drop/ Alter Index
* Show
* Describe

DML Statements

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

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

* To Create a database named “STUDENTS” with comments and database properties

hive> CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT ‘STUDENT Details’ WITH DBPROPERTIES (‘creator = “JOHN);

* To display a list of all databases

hive> SHOW DATABASES;

* To describe a database

Hive> DESCRIBE DATABASE STUDENTS;

* To alter the database properties

hive>ALTER DATABASE STUDENTS SET DBPROPERTIES (‘edited-by’= ‘JAMES’);

* To make the database as current working database

hive>USE STUDENTS;

* To drop database

hive>DROP DATABASE STUDENTS;

**Tables**

* Hive provides two kinds of tables

Managed and External Table

Managed Table –

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

* To create managed table named ‘STUDENT’

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

* To describe STUDENT Table

hive> DESCRIBE STUDENT;

Hive creates managed table in the warehouse directory of Hive .

External or Self Managed Tables

* When the table is dropped, it retains the data in the underlying location
* External keyword is used to create an external table
* Location needs to be specified to store the dataset in that particular location

Hive> CREATE EXTERNAL TABLE IF NOT EXISTS EXT\_STUDENT(rollno INT, name STRING, gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘\t’ LOCATION ‘/STUDENT\_INFO;

**Loading Data into Table from File**

* LOAD DATA LOCAL INPATH ‘/root/hivedemos/student.tsv’ OVERWRITE INTO TABLE EXT\_STUDENT;

Local keyword is used to load the data from local file system. To load the data from HDFS, remove local keyword from the statement.

Querying Table

* SELECT \* from EXT\_STUDENT;
* SELECT \* from STUDENT\_INFO;
* SELECT NAME , SUB FROM STUDENT\_INFO;

**Partitions**

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

Static partitions comprise columns whose values are known at compile time

* Dynamic Partition

Have columns whose values are known only at execution Time.

**Bucketing**

Bucketing is similar to partition.Howver there is subtle difference between partition and bucketing. In a partition, you need to create partition for each unique value of the column.This may lead to situations where you may end up with thousands of partitions.This can be avoided by using Bucketing in which you can limit the number of buckets to create.A Bucket is a file whereas partition is a directory.

**PIG**

## What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze large sets of data representing them as data flows. Pig is generally used with **Hadoop**; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

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. Apache Pig has a component known as **Pig Engine** that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

## Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

* Using **Pig Latin**, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
* Apache Pig uses **multi-query approach**, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
* Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.
* Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

## Features of Pig

Apache Pig comes with the following features −

* **Rich set of operators** − It provides many operators to perform operations like join, sort, filter, etc.
* **Ease of programming** − Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
* **Optimization opportunities** − The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
* **Extensibility** − Using the existing operators, users can develop their own functions to read, process, and write data.
* **UDF’s** − Pig provides the facility to create **User-defined Functions** in other programming languages such as Java and invoke or embed them in Pig Scripts.
* **Handles all kinds of data** − Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

## Apache Pig Vs MapReduce

Listed below are the major differences between Apache Pig and MapReduce.

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

## Apache Pig Vs SQL

Listed below are the major differences between Apache Pig and SQL.

|  |  |
| --- | --- |
| **Pig** | **SQL** |
| Pig Latin is a **procedural** language. | SQL is a **declarative** language. |
| In Apache Pig, **schema** is optional. We can store data without designing a schema (values are stored as $01, $02 etc.) | Schema is mandatory in SQL. |
| The data model in Apache Pig is **nested relational**. | The data model used in SQL **is flat relational**. |
| Apache Pig provides limited opportunity for **Query optimization**. | There is more opportunity for query optimization in SQL. |

In addition to above differences, Apache Pig Latin −

* Allows splits in the pipeline.
* Allows developers to store data anywhere in the pipeline.
* Declares execution plans.
* Provides operators to perform ETL (Extract, Transform, and Load) functions.

## Apache Pig Vs Hive

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

|  |  |
| --- | --- |
| **Apache Pig** | **Hive** |
| Apache Pig uses a language called **Pig Latin**. It was originally created at **Yahoo**. | Hive uses a language called **HiveQL**. It was originally created at **Facebook**. |
| Pig Latin is a data flow language. | HiveQL is a query processing language. |
| Pig Latin is a procedural language and it fits in pipeline paradigm. | HiveQL is a declarative language. |
| Apache Pig can handle structured, unstructured, and semi-structured data. | Hive is mostly for structured data. |

## Applications of Apache Pig

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used −

* To process huge data sources such as web logs.
* To perform data processing for search platforms.
* To process time sensitive data loads.

## Apache Pig – History

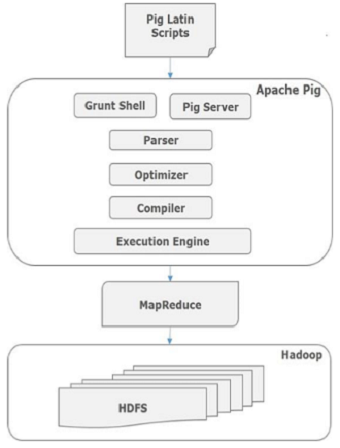
In **2006**, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset. In **2007**, Apache Pig was open sourced via Apache incubator. In **2008**, the first release of Apache Pig came out. In **2010**, Apache Pig graduated as an Apache top-level project.

# Apache Pig - Architecture

The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a high level data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer’s job easy. The architecture of Apache Pig is shown below.



## 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

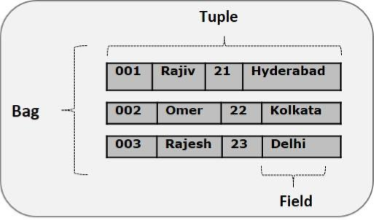
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

### 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

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**. Given below is the diagrammatic representation of Pig Latin’s data model.



### Atom

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. 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 (non-unique) 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, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

**Example** − {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**.

**Example** − {Raja, 30, **{9848022338, raja@gmail.com,}**}

### 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]

### Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

# Apache Pig - Grunt Shell

After invoking the Grunt shell, you can run your Pig scripts in the shell. In addition to that, there are certain useful shell and utility commands provided by the Grunt shell. This chapter explains the shell and utility commands provided by the Grunt shell.

**Note** − In some portions of this chapter, the commands like **Load** and **Store** are used. Refer the respective chapters to get in-detail information on them.

## Shell Commands

The Grunt shell of Apache Pig is mainly used to write Pig Latin scripts. Prior to that, we can invoke any shell commands using **sh** and **fs**.

### sh Command

Using **sh** command, we can invoke any shell commands from the Grunt shell. Using **sh** command from the Grunt shell, we cannot execute the commands that are a part of the shell environment (**ex** − cd).

**Syntax**

Given below is the syntax of **sh** command.

grunt> sh shell command parameters

## Pig Latin – Statements

While processing data using Pig Latin, **statements** are the basic constructs.

* Pig Latin statements are basic constructs to process data using Pig.Pig Latin statement is an operator.
* These statements work with **relations**. They include **expressions** and **schemas**.
* Every statement ends with a semicolon (;).
* We will perform various operations using operators provided by Pig Latin, through statements.
* Except LOAD and STORE, while performing all other operations, Pig Latin statements take a relation as input and produce another relation as output.
* As soon as you enter a **Load** statement in the Grunt shell, its semantic checking will be carried out. To see the contents of the schema, you need to use the **Dump** operator. Only after performing the **dump** operation, the MapReduce job for loading the data into the file system will be carried out.
* Pig Latin statements are generally ordered as follows:

1. LOAD statement that reads data from the file system.
2. Series of statements to perform transformations.
3. DUMP or STORE to display/store result.

The following is a simple Pig latin script to load, filter and store “student” data.

A = load ‘student’ (rollno, name, gpa);

A = filter A by gpa>4.0;

A= foreach A generate UPPER (name);

STORE A INTO ‘myreport’

In the above example A is a relation , not variable.

## Pig Latin – Data types

Given below table describes the Pig Latin data types.

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Data Type** | **Description & Example** |
| 1 | Int | Represents a signed 32-bit integer.  **Example** : 8 |
| 2 | Long | Represents a signed 64-bit integer.  **Example** : 5L |
| 3 | Float | Represents a signed 32-bit floating point.  **Example** : 5.5F |
| 4 | Double | Represents a 64-bit floating point.  **Example** : 10.5 |
| 5 | Chararray | Represents a character array (string) in Unicode UTF-8 format.  **Example** : ‘hadooppig’ |
| 6 | Bytearray | Represents a Byte array (blob). |
| 7 | Boolean | Represents a Boolean value.  **Example** : true/ false. |
| 8 | Datetime | Represents a date-time.  **Example** : 1970-01-01T00:00:00.000+00:00 |
| 9 | Biginteger | Represents a Java BigInteger.  **Example** : 60708090709 |
| 10 | Bigdecimal | Represents a Java BigDecimal  **Example** : 185.98376256272893883 |
| **Complex Types** | | |
| 11 | Tuple | A tuple is an ordered set of fields.  **Example** : (raja, 30) |
| 12 | Bag | A bag is a collection of tuples.  **Example** : {(raju,30),(Mohhammad,45)} |
| 13 | Map | A Map is a set of key-value pairs.  **Example** : [ ‘name’#’Raju’, ‘age’#30] |

## Null Values

Values for all the above data types can be NULL. Apache Pig treats null values in a similar way as SQL does.

A null can be an unknown value or a non-existent value. It is used as a placeholder for optional values. These nulls can occur naturally or can be the result of an operation.

## Pig Latin – Arithmetic Operators

The following table describes the arithmetic operators of Pig Latin. Suppose a = 10 and b = 20.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | **Addition** − Adds values on either side of the operator | a + b will give 30 |
| − | **Subtraction** − Subtracts right hand operand from left hand operand | a − b will give −10 |
| \* | **Multiplication** − Multiplies values on either side of the operator | a \* b will give 200 |
| / | **Division** − Divides left hand operand by right hand operand | b / a will give 2 |
| % | **Modulus** − Divides left hand operand by right hand operand and returns remainder | b % a will give 0 |
| ? : | **Bincond** − Evaluates the Boolean operators. It has three operands as shown below.  variable **x** = (expression) ? **value1** *if true* : **value2** *if false*. | b = (a == 1)? 20: 30;  if a = 1 the value of b is 20.  if a!=1 the value of b is 30. |
| CASE  WHEN  THEN  ELSE END | **Case** − The case operator is equivalent to nested bincond operator. | CASE f2 % 2  WHEN 0 THEN 'even'  WHEN 1 THEN 'odd'  END |

## Pig Latin – Comparison Operators

The following table describes the comparison operators of Pig Latin.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | **Equal** − Checks if the values of two operands are equal or not; if yes, then the condition becomes true. | (a = b) is not true |
| != | **Not Equal** − Checks if the values of two operands are equal or not. If the values are not equal, then condition becomes true. | (a != b) is true. |
| > | **Greater than** − Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition becomes true. | (a > b) is not true. |
| < | **Less than** − Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true. | (a < b) is true. |
| >= | **Greater than or equal to** − Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition becomes true. | (a >= b) is not true. |

|  |  |  |
| --- | --- | --- |
| <= | **Less than or equal to** − Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true. | (a <= b) is true. |
| matches | **Pattern matching** − Checks whether the string in the left-hand side matches with the constant in the right-hand side. | f1 matches '.\*tutorial.\*' |

## Pig Latin – Type Construction Operators

The following table describes the Type construction operators of Pig Latin.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| () | **Tuple constructor operator** − This operator is used to construct a tuple. | (Raju, 30) |
| {} | **Bag constructor operator** − This operator is used to construct a bag. | {(Raju, 30), (Mohammad, 45)} |
| [] | **Map constructor operator** − This operator is used to construct a tuple. | [name#Raja, age#30] |

## Pig Latin – Relational Operations

The following table describes the relational operators of Pig Latin.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **Loading and Storing** | |
| LOAD | To Load the data from the file system (local/HDFS) into a relation. |
| STORE | To save a relation to the file system (local/HDFS). |
| **Filtering** | |
| FILTER | To remove unwanted rows from a relation. |
| DISTINCT | To remove duplicate rows from a relation. |
| FOREACH, GENERATE | To generate data transformations based on columns of data. |
| STREAM | To transform a relation using an external program. |
| **Grouping and Joining** | |
| JOIN | To join two or more relations. |
| COGROUP | To group the data in two or more relations. |
| GROUP | To group the data in a single relation. |
| CROSS | To create the cross product of two or more relations. |
| **Sorting** | |
| ORDER | To arrange a relation in a sorted order based on one or more fields (ascending or descending). |
| LIMIT | To get a limited number of tuples from a relation. |
| **Combining and Splitting** | |
| UNION | To combine two or more relations into a single relation. |
| SPLIT | To split a single relation into two or more relations. |
| **Diagnostic Operators** | |
| DUMP | To print the contents of a relation on the console. |
| DESCRIBE | To describe the schema of a relation. |
| EXPLAIN | To view the logical, physical, or MapReduce execution plans to compute a relation. |
| ILLUSTRATE | To view the step-by-step execution of a series of statements. |

# Apache Pig - User Defined Functions

In addition to the built-in functions, Apache Pig provides extensive support for **U**ser **D**efined **F**unctions (UDF’s). Using these UDF’s, we can define our own functions and use them. The UDF support is provided in six programming languages, namely, Java, Jython, Python, JavaScript, Ruby and Groovy.

For writing UDF’s, complete support is provided in Java and limited support is provided in all the remaining languages. Using Java, you can write UDF’s involving all parts of the processing like data load/store, column transformation, and aggregation. Since Apache Pig has been written in Java, the UDF’s written using Java language work efficiently compared to other languages.

In Apache Pig, we also have a Java repository for UDF’s named **Piggybank**. Using Piggybank, we can access Java UDF’s written by other users, and contribute our own UDF’s.

## Types of UDF’s in Java

While writing UDF’s using Java, we can create and use the following three types of functions −

* **Filter Functions** − The filter functions are used as conditions in filter statements. These functions accept a Pig value as input and return a Boolean value.
* **Eval Functions** − The Eval functions are used in FOREACH-GENERATE statements. These functions accept a Pig value as input and return a Pig result.
* **Algebraic Functions** − The Algebraic functions act on inner bags in a FOREACHGENERATE statement. These functions are used to perform full MapReduce operations on an inner bag.