# Apache PIG

By-Shraddha Nikam

# What is ETL (Extract Transform Load)?

Extract, transform, and load (ETL) is the process of combining data from multiple sources into a large, central repository called a data warehouse.

ETL uses a set of business rules to clean and organize raw data and prepare it for storage, data analytics, and machine learning (ML).

You can address specific business intelligence needs through data analytics (such as predicting the outcome of business decisions, generating reports and dashboards, reducing operational inefficiency, and more).

# The ETL Process Explained

## Extract

Retrieves and verifies data from various sources

## Transform

Processes and organizes extracted data so it is usable

## Load

Moves transformed data to a data repository

# Why is ETL important?

Organizations today have both structured and unstructured data from various sources including:

- Customer data from online payment and customer relationship management (CRM) systems
- Inventory and operations data from vendor systems
- Sensor data from Internet of Things (IoT) devices
- Marketing data from social media and customer feedback
- Employee data from internal human resources systems

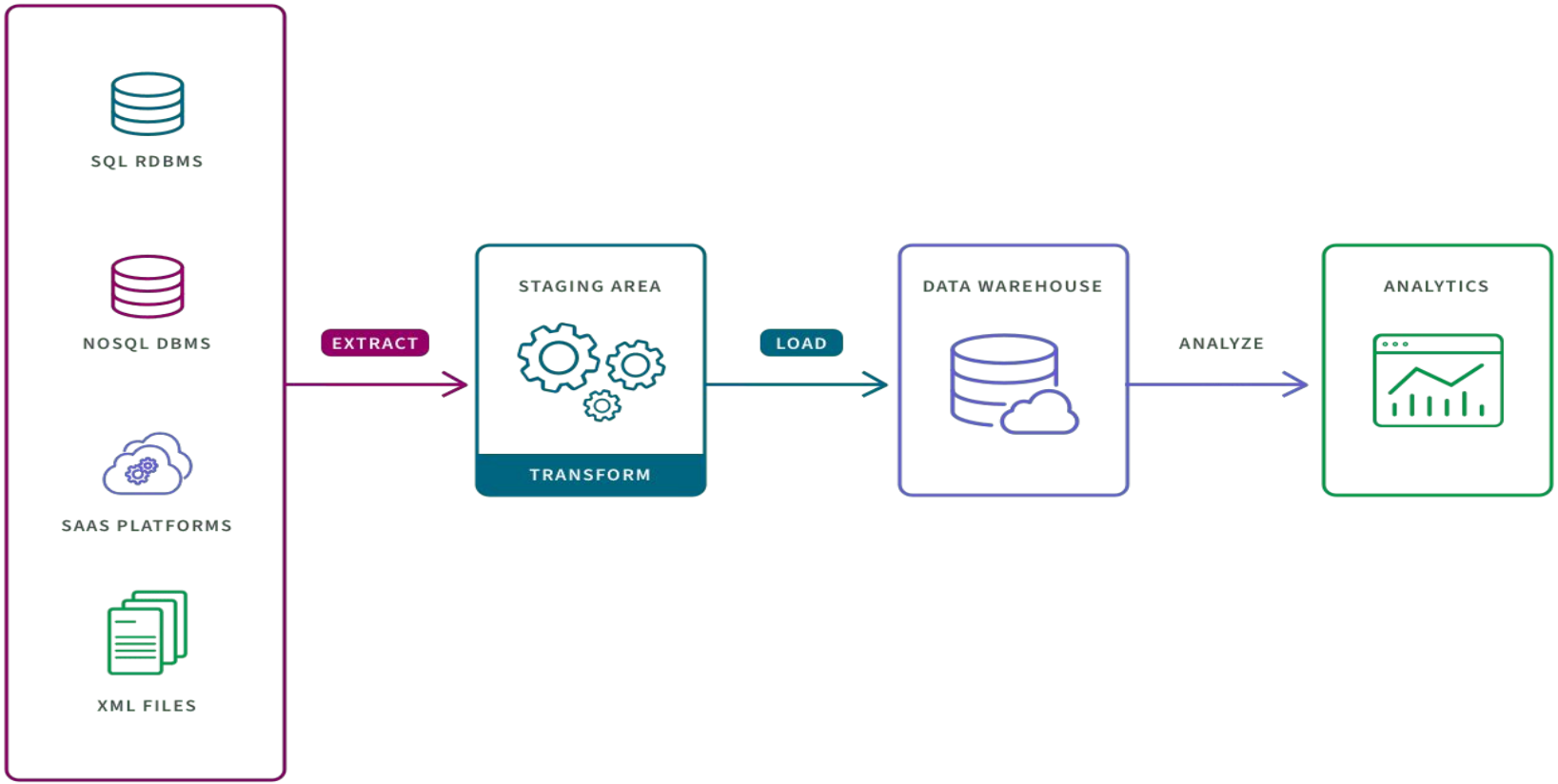
By applying the process of extract, transform, and load (ETL), individual raw datasets can be prepared in a format and structure that is more consumable for analytics purposes, resulting in more meaningful insights.

For example, online retailers can analyze data from points of sale to forecast demand and manage inventory. Marketing teams can integrate CRM data with customer feedback on social media to study consumer behavior.

# How does ETL work?

The traditional ETL process is broken out as follows:

- **Extract** refers to pulling a predetermined subset of data from a source such as an SQL or NoSQL database, a cloud platform or an XML file.
- **Transform** refers to converting the structure or format of a data set to match that of the target system. This is typically performed in a staging area in ways such as data mapping, applying concatenations or calculations. Transforming the data before it is loaded is necessary to deal with the constraints of traditional data warehouses.
- **Load** refers to the process of placing the data into the target system, typically a data warehouse, where it is ready to be analyzed by BI tools or data analytics tools.


SQL RDBMS
NOSQL DBMS
SAAS PLATFORMS
XML FILES
EXTRACT
STAGING AREA
TRANSFORM
LOAD
DATA WAREHOUSE
ANALYZE
ANALYTICS

# What is data Extraction?

In data extraction, extract, transform, and load (ETL) tools extract or copy raw data from multiple sources and store it in a staging area. A staging area (or landing zone) is an intermediate storage area for temporarily storing extracted data. Data staging areas are often transient, meaning their contents are erased after data extraction is complete. However, the staging area might also retain a data archive for troubleshooting purposes.

How frequently the system sends data from the data source to the target data store depends on the underlying change data capture mechanism. Data extraction commonly happens in one of the three following ways.

**Update notification:** In update notification, the source system notifies you when a data record changes. You can then run the extraction process for that change. Most databases and web applications provide update mechanisms to support this data integration method.

**Incremental extraction:** Some data sources can't provide update notifications but can identify and extract data that has been modified over a given time period. In this case, the system checks for changes at periodic intervals, such as once a week, once a month, or at the end of a campaign. You only need to extract data that has changed.

**Full extraction:** Some systems can't identify data changes or give notifications, so reloading all data is the only option. This extraction method requires you to keep a copy of the last extract to check which records are new. Because this approach involves high data transfer volumes, we recommend you use it only for small tables.

# What is data Transformation?

In data transformation, extract, transform, and load (ETL) tools transform and consolidate the raw data in the staging area to prepare it for the target data warehouse. The data transformation phase can involve the following types of data changes.

**Basic data transformation**

Basic transformations improve data quality by removing errors, emptying data fields, or simplifying data. Examples of these transformations follow.

***Data cleansing:*** Data cleansing removes errors and maps source data to the target data format. For example, you can map empty data fields to the number 0, map the data value "Parent" to "P," or map "Child" to "C."

***Data deduplication:*** Deduplication in data cleansing identifies and removes duplicate records.

***Data format revision :*** Format revision converts data, such as character sets, measurement units, and date/time values, into a consistent format. For example, a food company might have different recipe databases with ingredients measured in kilograms and pounds. ETL will convert everything to pounds.

## Advanced data transformation

Advanced transformations use business rules to optimize the data for easier analysis. Examples of these transformations follow.

***Derivation:*** Derivation applies business rules to your data to calculate new values from existing values. For example, you can convert revenue to profit by subtracting expenses or calculating the total cost of a purchase by multiplying the price of each item by the number of items ordered.

***Joining:*** In data preparation, joining links the same data from different data sources. For example, you can find the total purchase cost of one item by adding the purchase value from different vendors and storing only the final total in the target system.

***Splitting:*** You can divide a column or data attribute into multiple columns in the target system. For example, if the data source saves the customer name as “Jane John Doe,” you can split it into a first, middle, and last name.

***Summarization:*** Summarization improves data quality by reducing a large number of data values into a smaller dataset. For example, customer order invoice values can have many different small amounts. You can summarize the data by adding them up over a given period to build a customer lifetime value (CLV) metric.

***Encryption:*** You can protect sensitive data to comply with data laws or data privacy by adding encryption before the data streams to the target database.

# What is data Loading?

In data loading, extract transform, and load (ETL) tools move the transformed data from the staging area into the target data warehouse. For most organizations that use ETL, the process is automated, well defined, continual, and batch driven. Two methods for loading data follow.

**Full load:** In full load, the entire data from the source is transformed and moved to the data warehouse. The full load usually takes place the first time you load data from a source system into the data warehouse.

**Incremental load :**In incremental load, the ETL tool loads the delta (or difference) between target and source systems at regular intervals. It stores the last extract date so that only records added after this date are loaded. There are two ways to implement incremental load.

***Streaming incremental load :*** If you have small data volumes, you can stream continual changes over data pipelines to the target data warehouse. When the speed of data increases to millions of events per second, you can use event stream processing to monitor and process the data streams to make more-timely decisions.

***Batch incremental load:*** If you have large data volumes, you can collect load data changes into batches periodically. During this set period of time, no actions can happen to either the source or target system as data is synchronized.

# Introduction to Apache Pig

Pig Represents Big Data as data flows. Pig is a high-level platform or tool which is used to process the large datasets. It provides a high-level of abstraction for processing over the MapReduce. It provides a high-level scripting language, known as *Pig Latin* which is used to develop the data analysis codes.

First, to process the data which is stored in the HDFS, the programmers will write the scripts using the Pig Latin Language. Internally *Pig Engine*(a component of Apache Pig) converted all these scripts into a specific map and reduce task. But these are not visible to the programmers in order to provide a high-level of abstraction. Pig Latin and Pig Engine are the two main components of the Apache Pig tool. The result of Pig always stored in the HDFS.

# Need of Pig:

One limitation of MapReduce is that the development cycle is very long. Writing the reducer and mapper, compiling packaging the code, submitting the job and retrieving the output is a time-consuming task. Apache Pig reduces the time of development using the multi-query approach. Also, Pig is beneficial for programmers who are not from Java background. 200 lines of Java code can be written in only 10 lines using the Pig Latin language. Programmers who have SQL knowledge needed less effort to learn Pig Latin.

- It uses query approach which results in reducing the length of the code.
- Pig Latin is SQL like language.
- It provides many builtIn operators.
- It provides nested data types (tuples, bags, map).

# Apache Pig Execution Modes

We can start Apache Pig in two modes, the first mode is Local and the second mode is Mapreduce or HDFS.

## 1. Local Mode

In this mode of execution, we need a single machine and all files are installed and run using your localhost and file system. This mode is used for testing and development purposes. The local mode does not need HDFS or Hadoop.

To start Local mode type the below command.

**$pig -x local**

## 2. Mapreduce Mode

Mapreduce is the default mode of the Apache Pig Grunt shell. In this mode, we need to load data in HDFS and then we can perform the operation. When we run the Pig Latin command on that data, a MapReduce job is started in the back-end to operate.

To start the Mapreduce mode type the below command.

**$pig -x mapreduce**
or
**$pig**

# Pig vs. SQL

| Difference | Pig | SQL |
|---|---|---|
| **Definition** | Pig is a scripting language used to interact with HDFS. | SQL is a query language used to interact with databases residing in the database engine. |
| **Query Style** | Pig offers a step-by-step execution style. | SQL offers the single block execution style. |
| **Evaluation** | Pig does a lazy evaluation, which means that data is processed only when the STORE or DUMP command is encountered. | SQL offers immediate evaluation of a query. |
| **Pipeline Splits** | Pipeline Splits are supported in Pig. | In SQL, you need to run the “join” command twice for the result to be materialized as an intermediate result. |

# NoSQL Databases

These databases break one or more of the traditional rules of relational database systems. They do not expect data to be normalized. Instead, the data accessed by a single application lives in one large table so that few or no joins are necessary. Many of these databases do not implement full ACID semantics.

Like MapReduce, these systems are built to manage terabytes of data. Unlike MapReduce, they are focused on random reads and writes of data. Where MapReduce and technologies built on top of it (such as Pig) are optimized for reading vast quantities of data very quickly, these NoSQL systems optimize for finding a few records very quickly. This different focus does not mean that Pig does not work with these systems. Users often want to analyze the data stored in these systems. Also, because these systems offer good random lookup, certain types of joins could benefit from having the data stored in these systems.

Two NoSQL databases have been integrated with Pig: HBase and Cassandra.

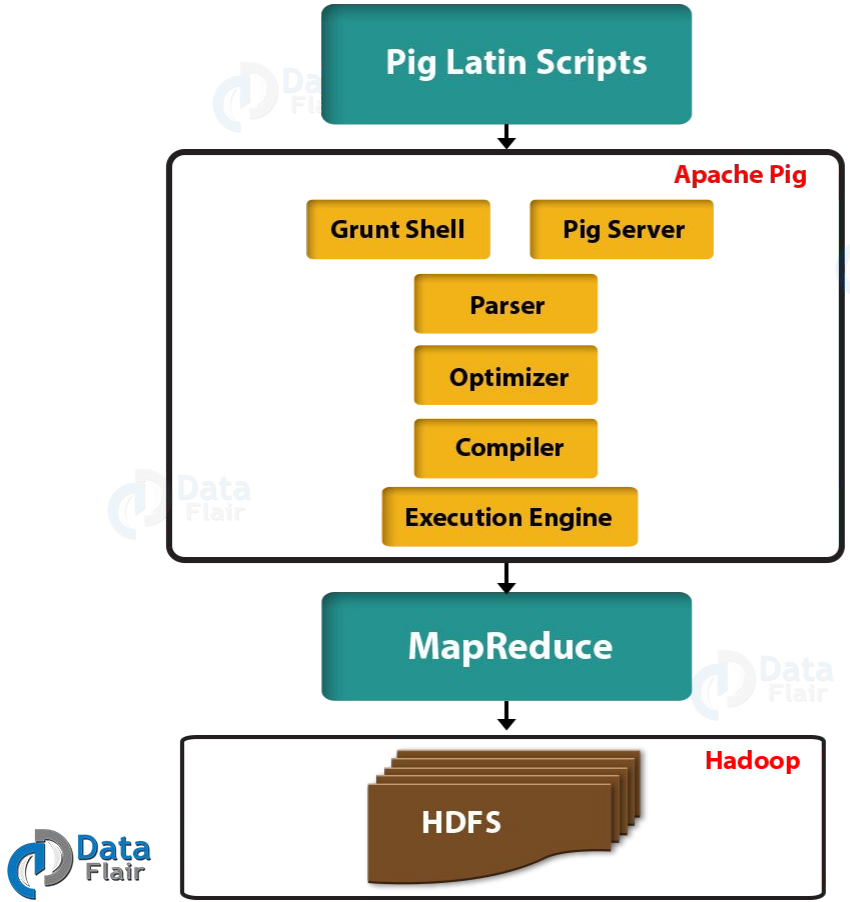
# Apache Pig - Architecture

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.

# Architecture of Apache Pig



# 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 data type

Pig data type can be classified into two categories, and they are −

- Primitive
- Complex

**Primitive Data type**

It is also named as Simple Data type. The primitive data types are as follows −

| Data 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 world |
| byteArray | Byte array | byte[] |

## Complex Data type

Complex data types consist of a bit of logical and complicated data type. The following are the complex data type −

| Data Types | Definition | Code | Example |
|---|---|---|---|
| **Tuple** | A set of ordered fields. The tuple is written with braces. | (field[,fields....]) | (1,2) |
| **Bag** | A group of tuples is called a bag. Represented by folded weights or curly braces. | {tuple,[,tuple...]} | {(1,2), (3,4)} |
| **Map** | A set of key-value pairs. The map is represented by square brackets. | [Key # Value] | ['keyname'#'valuename'] |

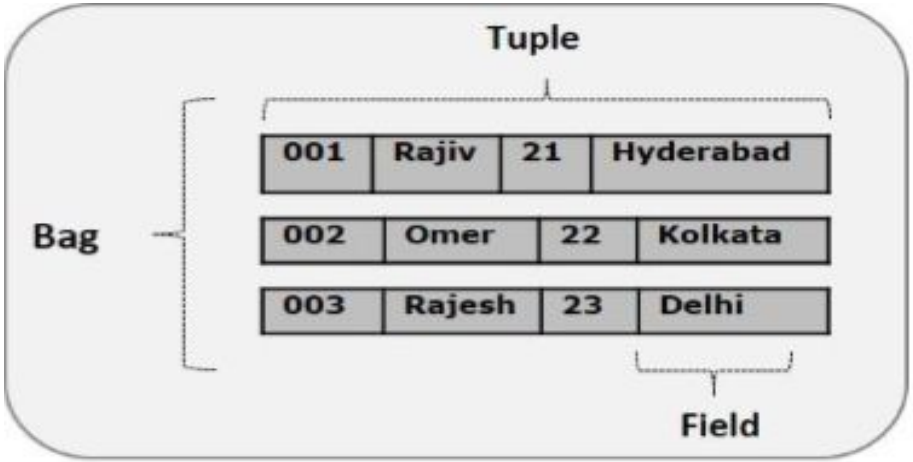
- **Key** − An element of finding an element, the key must be unique and must be charrarray.
- **Value** − Any data can be stored in a value, and each key has particular data related to it. The map is built using a bracket and hash between key and values. Cas to separate pairs of over one key value. Here # is used to distinguish key and value.
- **Null Values** − Valuable value is missing or unknown, and any data may apply. The pig handles an empty value similar to SQL. Pig detects blank values when data is missing, or an error occurs during data processing. Also, null can be used as a value proposition of your choice.

# Pig Latin Data Model

- Atom
- Tuple
- Bag
- Map
- Relation

# Pig Data Model

Pig Latin is the language used to analyze data in Hadoop using Apache Pig.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.



**1. 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’

**2. 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)

**3. 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,}**}

**4. 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]

**5. 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

Apache Pig Grunt is an interactive shell that enables users to enter Pig Latin interactively and provides a shell to interact with HDFS and local file system commands. You can enter Pig Latin commands directly into the Grunt shell for execution. Apache Pig starts executing the Pig Latin language when it receives the STORE or DUMP command. Before executing the command Pig Grunt shell do check the syntax and semantics to void any error.

To start Pig Grunt type : **$pig -x local**

It will start Pig Grunt shell: **grunt>**

Now using Grunt shell you can interact with your local filesystem. But if you forget the -x local and have a cluster configuration set in PIG_CLASSPATH, then it put you in a Grunt shell that will interact with HDFS on your cluster.

# 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**.

## 1. 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

**Example:**

We can invoke the **ls** command of Linux shell from the Grunt shell using the **sh** option as shown below. In this example, it lists out the files in the **/pig/bin/** directory.

**grunt> sh ls**

pig

pig_1444799121955.log

pig.cmd

pig.py

## 2. fs Command

Using the **fs** command, we can invoke any FsShell commands from the Grunt shell.

**Syntax**

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

grunt> sh File System command parameters

**Example**

We can invoke the ls command of HDFS from the Grunt shell using fs command. In the following example, it lists the files in the HDFS root directory.

**grunt> fs –ls**

Found 3 items

drwxrwxrwx - Hadoop supergroup 0 2015-09-08 14:13 Hbase

drwxr-xr-x - Hadoop supergroup 0 2015-09-09 14:52 seqgen_data

drwxr-xr-x - Hadoop supergroup 0 2015-09-08 11:30 twitter_data

In the same way, we can invoke all the other file system shell commands from the Grunt shell using the **fs** command.

# Utility Commands

The Grunt shell provides a set of utility commands. These include utility commands such as **clear, help, history, quit,** and **set**; and commands such as **exec, kill,** and **run** to control Pig from the Grunt shell. Given below is the description of the utility commands provided by the Grunt shell.

## 1. clear Command

The **clear** command is used to clear the screen of the Grunt shell.

**Syntax**

You can clear the screen of the grunt shell using the **clear** command as shown below.

grunt> clear

## 2. help Command

The **help** command gives you a list of Pig commands or Pig properties.

### Usage

You can get a list of Pig commands using the **help** command as shown below.

**grunt> help**

Commands: <pig latin statement>; - See the PigLatin manual for details:

http://hadoop.apache.org/pig

### 3. history Command

This command displays a list of statements executed / used so far since the Grunt sell is invoked.

### Usage

Assume we have executed three statements since opening the Grunt shell.

```
grunt> customers = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING PigStorage(',');

grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING PigStorage(',');

grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING PigStorage(',');
```

Then, using the **history** command will produce the following output.

**grunt> history**

customers = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING PigStorage(',');

orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING PigStorage(',');

student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING PigStorage(',');

## 4. Set Command

The SET command is used to assign values to keys that are case sensitive. In case the SET command is used without providing arguments then all other system properties and configurations are printed.

**Syntax:**

**grunt> set [key 'value']**

**Command:**

**grunt> SET debug 'on'**
**grunt> SET job.name 'my job'**

**grunt> SET default_parallel 100**

Using this command, you can set values to the following keys.

| Key | Description |
| --- | --- |
| **default_parallel** | Using this parameter you can set the number of reducers for all MapReduce jobs generated by Pig. |
| **debug** | Using this parameter you can turn debug-level logging on or off. |
| **job.name** | Using this parameter you can set a user-specified name for the job. |
| **job.priority** | Using this parameter you can set the priority of a Pig job such as very_low, low, normal, high, very_high. |
| **stream.skippath** | Using this parameter you can set the path from where the data is not to be transferred, bypassing the desired path in the form of a string to this key. |

## 5. quit Command

You can quit from the Grunt shell using this command.

**Usage**

Quit from the Grunt shell as shown below.

**grunt> quit**

Let us now take a look at the commands using which you can control Apache Pig from the Grunt shell.

1. **exec Command**

Using the **exec** command, we can execute Pig scripts from the Grunt shell.

**Syntax**

Given below is the syntax of the utility command **exec**.

**grunt> exec [–param param_name = param_value] [–param_file file_name] [script]**

**Example**

Let us assume there is a file named **student.txt** in the **/pig_data/** directory of HDFS with the following content.

**Student.txt**

001,Rajiv,Hyderabad

002,siddarth,Kolkata

003,Rajesh,Delhi

And, assume we have a script file named **sample_script.pig** in the **/pig_data/** directory of HDFS with the following content.

**Sample_script.pig**

```
student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING PigStorage(',')
      as (id:int,name:chararray,city:chararray);
Dump student;
```

Now, let us execute the above script from the Grunt shell using the **exec** command as shown below.

**grunt> exec /sample_script.pig**

**Output**

The **exec** command executes the script in the **sample_script.pig**. As directed in the script, it loads the **student.txt** file into Pig and gives you the result of the Dump operator displaying the following content.

(1,Rajiv,Hyderabad)

(2,siddarth,Kolkata)

(3,Rajesh,Delhi)

## 2. kill Command

You can kill a job from the Grunt shell using this command.

### Syntax

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

**grunt> kill JobId**

### Example

Suppose there is a running Pig job having id **Id_0055**, you can kill it from the Grunt shell using the **kill** command, as shown below.

**grunt> kill Id_0055**

**3. run Command**

You can run a Pig script from the Grunt shell using the **run** command

**Syntax :** Given below is the syntax of the **run** command.

**grunt> run [–param param_name = param_value] [–param_file file_name] script**

**Example :** Let us assume there is a file named **student.txt** in the **/pig_data/** directory of HDFS with the following content.

**Student.txt**

001,Rajiv,Hyderabad

002,siddarth,Kolkata

003,Rajesh,Delhi

And, assume we have a script file named **sample_script.pig** in the local filesystem with the following content.

**Sample_script.pig**

```
student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USINGPigStorage(',')
      as (id:int,name:chararray,city:chararray);
```

Now, let us run the above script from the Grunt shell using the run command as shown below.

**grunt> run /sample_script.pig**

You can see the output of the script using the **Dump operator** as shown below.

**grunt> Dump;**

(1,Rajiv,Hyderabad)

(2,siddarth,Kolkata)

(3,Rajesh,Delhi)

**Note** − The difference between **exec** and the **run** command is that if we use **run**, the statements from the script are available in the command history.

# 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.