# Apache Pig

**1: Introducing Apache Pig**

* Apache Pig is a **high-level platform** for processing large datasets
* Uses a scripting language called **Pig Latin**
* Designed to simplify **MapReduce programming** on Hadoop
* Ideal for **ETL tasks**, data transformation, and analysis
* Developed by Yahoo! and runs on the **Hadoop ecosystem**
* Can process both **structured and semi-structured data**
* Great for data engineers who prefer scripts over Java

### 2: ****The Pig Architecture****

* Pig has **two main components**:
  + **Pig Latin Scripts** (user-written code)
  + **Execution Engine** (runs on Hadoop)
* Scripts are parsed and converted into **logical plans**
* Logical plans are then turned into **physical plans**
* Physical plans are translated into **MapReduce jobs**
* Pig uses **Grunt Shell** for interactive execution
* Works with **HDFS** to read/write large datasets
* Supports both **local mode** and **MapReduce mode**

### 3: ****Benefits of Apache Pig****

* **Simplifies complex data processing** without writing Java
* Uses **Pig Latin**, which is easy to learn and use
* Supports **parallel data processing** via Hadoop
* Great for **ETL tasks**, data cleaning, and transformations
* Flexible – handles **structured, semi-structured, and unstructured data**
* Enables rapid **development and prototyping**
* Easily integrates with **HDFS, HBase, Hive, and UDFs**

### 4: ****Properties of Pig****

* **Procedural Language**: Executes step-by-step, like a script
* **Lazy Evaluation**: Commands are not executed until a result is requested
* **Extensibility**: Supports **User Defined Functions (UDFs)** in Java, Python, etc.
* **Optimization**: Pig automatically optimizes execution plans
* **Schema-on-read**: Data doesn’t need to follow a fixed schema beforehand
* **Multi-language support**: UDFs can be written in multiple languages
* **Handles any data**: Works well with structured & unstructured data

**5: Running Pig**

* Pig can be run in **two modes**:
  + **Local Mode**: Runs on a single machine (for small data/testing)
  + **MapReduce Mode**: Runs on a Hadoop cluster (for big data)
* Entry point is the **Grunt Shell** (interactive command line)
* Scripts can also be run in **batch mode** using:

pig script.pig

* Input/output files are read/written from **HDFS or local file system**
* Supports parameter substitution using **-param or -param\_file**

### 6: ****Getting Started with Pig Latin****

* **Pig Latin** is a simple scripting language for data flows
* Works with **relations** (similar to tables)
* Common steps:
  + **LOAD** data from HDFS
  + **FILTER**, **FOREACH**, **GROUP**, **JOIN**, etc.
  + **DUMP** or **STORE** results
* Example:

data = LOAD 'input.txt' AS (name:chararray, age:int);

adults = FILTER data BY age > 18;

DUMP adults;

* Case-insensitive and easy to understand

### 7: ****Working with Operators in Pig****

* **LOAD**: Load data into a relation
* **STORE**: Save results to HDFS
* **FILTER**: Select tuples based on conditions
* **FOREACH...GENERATE**: Transform data or create new fields
* **GROUP**: Group data by a field for aggregation
* **JOIN**: Combine data from two or more relations
* **ORDER**: Sort data by specified fields
* **DISTINCT**: Remove duplicate tuples
* Operators can be **chained** to build complex data flows

### 8: ****Working with Functions in Pig****

* Pig provides **built-in functions** for data manipulation:
  + String functions: UPPER(), LOWER(), SUBSTRING()
  + Mathematical functions: ABS(), FLOOR(), CEIL()
  + Date functions: CURRENT\_TIME(), TO\_DATE()
* Users can create **User Defined Functions (UDFs)** in Java, Python, etc.
* Functions can be used inside **FOREACH**, **FILTER**, and other operators
* Supports **aggregate functions** like SUM(), AVG(), COUNT()

### 9: ****Error Handling in Pig****

* Pig provides **error messages** to help debug scripts
* Common errors: syntax errors, data type mismatches, missing files
* Use **ILLUSTRATE** command to see how data flows through a script
* Check **log files** in Hadoop for detailed error info
* Use **TRY and CATCH** in UDFs to handle exceptions gracefully
* Best practice: test scripts on small data before running large jobs
* Use **comments** (-- or /\* \*/) for clarity and easier debugging

### Practical question:

1. Implement the following operations using Pig Operators/Functions: LOAD, DUMP, FOREACH, GROUP, DISTINCT, LIMIT, ORDER BY, JOIN, UNION, SPLIT, SAMPLE, AVG, MAX, COUNT, TUPLE, MAP, PIGGY BANK, PARAMETER SUBSTITUTION, DESCRIBE
2. Write a PIG script to load and store “Student data”. (Student file contains Roll no, Name, Marks and GPA).
   1. Filter all the students who are having GPA>5.
   2. Display the name of all Students in Uppercase.
   3. Group tuples of students based on their GPA.
   4. Remove duplicate tuples of Student list.
   5. Display first three tuples from “student” relation.
   6. Display the names of students in ascending order.
   7. Join two relations namely Student and department (Rno, DeptNo, DeptName) based on the values contained in the roll no column.
   8. Merge content of two relations Student and department.
   9. Partition a relation based on the GPA’s acquired by students.

**Part 1: Pig Operators/Functions Overview (Brief usage)**

| **Operator/Function** | **Purpose** | **Example Syntax** |
| --- | --- | --- |
| LOAD | Load data into a relation | students = LOAD 'data.txt'; |
| DUMP | Display contents of a relation | DUMP students; |
| FOREACH | Process/transform data | names = FOREACH students GENERATE name; |
| GROUP | Group data by key | grp = GROUP students BY GPA; |
| DISTINCT | Remove duplicates | unique = DISTINCT students; |
| LIMIT | Get first N rows | first3 = LIMIT students 3; |
| ORDER BY | Sort data | sorted = ORDER students BY name ASC; |
| JOIN | Join two relations | joined = JOIN students BY rollno, dept BY rollno; |
| UNION | Combine two relations | merged = UNION students, dept; |
| SPLIT | Split relation based on condition | SPLIT students INTO highGPA IF GPA > 5, lowGPA IF GPA <= 5; |
| SAMPLE | Take random sample | sampled = SAMPLE students 0.1; |
| AVG | Average of a group | avgGPA = FOREACH grp GENERATE AVG(students.GPA); |
| MAX | Maximum value | maxMarks = FOREACH grp GENERATE MAX(students.Marks); |
| COUNT | Count tuples | countStud = FOREACH grp GENERATE COUNT(students); |
| TUPLE | Tuple structure (row) | Pig data is stored as tuples |
| MAP | Key-value data structure | Maps store key-value pairs |
| PIGGY BANK | Library of UDFs | Use extra functions from PiggyBank |
| PARAMETER SUBSTITUTION | Pass parameters to scripts | pig -param GPA=5 script.pig |
| DESCRIBE | Show schema of relation | DESCRIBE students; |

**Part 2: Sample Pig Script for Student Data Tasks**

-- Load Student data (RollNo:int, Name:chararray, Marks:int, GPA:double)

students = LOAD 'student\_data.txt' USING PigStorage(',') AS (rollno:int, name:chararray, marks:int, gpa:double);

-- a) Filter students with GPA > 5

highGPA\_students = FILTER students BY gpa > 5;

-- b) Display student names in uppercase

upper\_names = FOREACH students GENERATE rollno, UPPER(name) AS name\_upper, marks, gpa;

-- c) Group students by GPA

grp\_by\_gpa = GROUP students BY gpa;

-- d) Remove duplicate tuples

unique\_students = DISTINCT students;

-- e) Display first 3 tuples

first\_three = LIMIT students 3;

-- f) Order students by name ascending

sorted\_by\_name = ORDER students BY name ASC;

-- Load department data (RollNo:int, DeptNo:int, DeptName:chararray)

department = LOAD 'department\_data.txt' USING PigStorage(',') AS (rollno:int, deptno:int, deptname:chararray);

-- g) Join students and department by RollNo

joined\_data = JOIN students BY rollno, department BY rollno;

-- h) Union students and department relations (assuming compatible schema or for demo)

-- For demo, let's union student rollno & name with department rollno & deptname as (rollno, info)

stud\_simple = FOREACH students GENERATE rollno, name AS info;

dept\_simple = FOREACH department GENERATE rollno, deptname AS info;

merged\_data = UNION stud\_simple, dept\_simple;

-- i) Split students by GPA partitions

SPLIT students INTO

highGPA IF gpa > 7,

midGPA IF gpa > 5 AND gpa <= 7,

lowGPA IF gpa <= 5;

-- Store or Dump results as needed

DUMP highGPA\_students;

### Stepwise Explanation for Pig Practical Operations

#### 1. **Loading Data**

students = LOAD 'student\_data.txt' USING PigStorage(',') AS (rollno:int, name:chararray, marks:int, gpa:double);

* Loads student data from a CSV file.
* Defines schema with Roll No, Name, Marks, GPA.
* PigStorage(',') tells Pig data is comma-separated.

#### 2. **Filter students with GPA > 5**

highGPA\_students = FILTER students BY gpa > 5;

* Filters tuples where GPA is greater than 5.
* Creates a new relation with only those students.

#### 3. **Convert student names to uppercase**

upper\_names = FOREACH students GENERATE rollno, UPPER(name) AS name\_upper, marks, gpa;

* Transforms each student’s name to uppercase using UPPER() function.
* Keeps other fields unchanged.

#### 4. **Group students by GPA**

grp\_by\_gpa = GROUP students BY gpa;

* Groups student records sharing the same GPA together.
* Useful for aggregate operations (like AVG, COUNT).

#### 5. **Remove duplicate student records**

unique\_students = DISTINCT students;

* Removes any duplicate tuples in the students relation.

#### 6. **Get first 3 student records**

first\_three = LIMIT students 3;

* Selects the first 3 rows from the students relation.

#### 7. **Sort students by name ascending**

sorted\_by\_name = ORDER students BY name ASC;

* Sorts the student list in alphabetical order by their names.

#### 8. **Load department data**

department = LOAD 'department\_data.txt' USING PigStorage(',') AS (rollno:int, deptno:int, deptname:chararray);

* Loads department info with roll no, department number, and name.

#### 9. **Join students and departments by Roll No**

joined\_data = JOIN students BY rollno, department BY rollno;

* Combines student and department data where roll numbers match.
* Creates a combined relation with fields from both.

#### 10. **Union (merge) students and department info**

stud\_simple = FOREACH students GENERATE rollno, name AS info;

dept\_simple = FOREACH department GENERATE rollno, deptname AS info;

merged\_data = UNION stud\_simple, dept\_simple;

* Transforms student and department data into a common format.
* Combines (unions) both datasets into one relation.

#### 11. **Split students based on GPA ranges**

SPLIT students INTO

highGPA IF gpa > 7,

midGPA IF gpa > 5 AND gpa <= 7,

lowGPA IF gpa <= 5;

* Divides the students relation into three parts based on GPA value.
* Helps analyze or process each partition separately.

#### 12. **Displaying or Saving Results**

DUMP highGPA\_students;

* Displays the filtered high GPA students on the console.