SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

Subject Name: BIG DATA ANALYTICS

Year / Branch: II MCA – I SEMESTER

Regulation: R24

Prepared By: Mr. N P Gangadhar,

Assistant Professor

II MCA – I SEMESTER			
COURSE CODE:	**24MCA213**	**CREDITS:**	**4**
COURSE TITLE:	**BIG DATA ANALYTICS**	**L-T-P:**	**4-0-0**

PREREQUISITES: *Courses on "Database Management Systems" , "Object Oriented Programming through JAVA" and knowledge on Intelligence Techniques.*

COURSE EDUCATIONAL OBJECTIVES*:*

CEO1 *To explore the fundamental concepts of Big Data.*
CEO2 *To Learn Basic concepts of Hadoop.*
CEO3 *To Write Hadoop MapReduce Programs for analyzing Big data.*
CEO4 *To Explore Hadoop Ecosystem.*

UNIT - I: UNDERSTANDING BIG DATA — **Lecture Hrs:10**

What is Big Data, Concepts and Terminology - Datasets, Data Analysis, Data Analytics, Big Data Characteristics – volume, velocity, variety, veracity, value, Different Types of Data – Structured Data, Unstructured Data, Semi-Structured Data, Case Study Background.

UNIT - II : HADOOP BASICS — **Lecture Hrs:10**

Brief history of Hadoop, Apache hadoop and the hadoop ecosystem. A weather dataset, analyzing the data with unix tools, analyzing the data with hadoop , Understanding different Hadoop modes, understanding Hadoop Features-Understanding HDFS, Understanding MapReduce, Learning the HDFS and Mapreduce Architecture-Understanding the HDFS architecture, Understanding the MapReduce Architecture, Understanding the HDFS and MapReduce architecture by plot.

UNIT - III: WORKING WITH PIG AND HIVE — **Lecture Hrs:12**

Pig -Execution Types, An Example, Pig Latin-Structure, Statements,Types, Schemas, Functions, Data Processing Operators.Hive – An example, Tables –Managed Tables and External tables, Partitions and Buckets, Importing data, Altering Data, Dropping Tables, Querying Data-Sorting and Aggragating, Mapreduce Scripts.

UNIT - IV : INTRODUCTION TO APACHE SPARK — **Lecture Hrs:12**

What is Apache Spark, Characteristics – Speed, Easy of Use, Modularity, Extensibility, Apache Spark Components as a Unified Stack, Spark Basic Data Types, Spark Core and SQL- Dynamic Partition Pruning, SQL Operations, Schemas and Creating Data Frames

UNIT - V: HBASE, ZOOKEEPER, SQOOP — **Lecture Hrs:12**

HBase Overview – Limitations of Hadoop, what is HBase,HBase and HDFS,StorageMechansim in HBase,Features of HBase, Applications of HBase. ZooKeeper Overview – what is ZooKeeper, Distributed Application, Benefits of Distributed Applications,Challenges of Distributed Applications, What is Apache Zookeeper meant for, Benefits of ZooKeeper

TEXT BOOKS:

1. *Big Data Fundamentals: Concepts, Drivers & Techniques", 1/e, 2016, Thomas Erl, Wajid Khattak, Paul Buhler, Prentice Hall.*
2. *"Hadoop:The Definitive Guide," 3/e, 2012, Tom White, O'REILLY Publications.*
3. *"Learning Spark"*

REFERENCE BOOKS:

1. *Taming the Big Data Tidal Wave: Finding Opportunities in Huge Data Streams with Advanced Analytics , 2012, Bill Franks, John Wiley & Sons..*
2. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, 2012, Paul C. Zikopoulos, Chris Eaton, Dirk deRoos, Thomas Deutsch, George Lapis, McGraw-Hill.*

3. *Intelligent Data Analysis, 2007, Michael Berthhold, David J.Hand, Springer.*

4. *Understanding Big Data: Analytics for Enterprise Class Hadoop and streaming data, 2011, Paul Zikopoulos, Chris Eaton, Paul Zikopoulos, McGraw hill.*

5. *Big Data for Dummies, 2012, Hurwitz, Alan Nugent, Dr. Fern Halper, Marcia Kaufman,John Wiley & Sons*

COURSE OUTCOMES: *On successful completion of this course, students will be able to:*		POs related to COs
CO1	**Realize** characteristics of Big Data and various types of data like structured, unstructured and semistructured	PO1,PO2
CO2	**Understand** two major components of Hadoop	PO1,PO2,PO3,PO4, PO5
CO3	**Analyze** Big data using Hadoop Map Reduce programs	PO1, PO2,PO4, PO5
CO4	get **Acquainted** with two Data Access Components of Hadoop Ecosystem called pig and hive	PO1,PO3, PO4
CO5	Acquire **Knowledge** on Data Storage Component of Hadoop Ecosystem called Hbase, Data Integration Component of Hadoop Ecosystem like sqoop and Monitoring, Management and Orchestration of Hadoop Ecosystem component like zookeeper	PO1,PO2, PO3,PO4

CO-PO MAPPING(DETAILED; HIGH:3; MEDIUM:2; LOW:1)

Course	POs / COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8
C303: Big Data Analytics	C303.1	3	2	-	-	-	-	-	-
	C303.2	3	3	2	3	2	-	-	-
	C303.3	3	3	-	3	3	-	-	-
	C303.4	3	-	3	3	-	-	-	-
	C303.5	3	3	2	3	-	-	-	-
	C303	3	2.75	2.3	3	2.5	-	-	-

UNIT III - WORKING WITH PIG AND HIVE

"Hive brings the power of SQL to the Hadoop ecosystem, while Pig simplifies complex data processing with an easy scripting language."

Unit Overview

This unit introduces **Apache Pig** and **Apache Hive**, two important tools in the Hadoop ecosystem used for processing and analyzing large datasets. Students will learn Pig execution types, Pig Latin language, data types, schemas, functions, and data processing operators. The unit also covers Hive architecture, managed and external tables, partitions, buckets, data import, table modifications, querying techniques, sorting, aggregation, and writing MapReduce scripts. These tools simplify Big Data analysis by providing high-level programming languages instead of writing complex MapReduce programs.

Learning Outcomes

After completing this unit, students will be able to:

1. Explain the purpose and architecture of Apache Pig.
2. Differentiate between Pig execution modes.
3. Write Pig Latin programs for data processing.
4. Explain Pig data types, schemas, statements, and functions.
5. Use Pig operators for filtering, grouping, joining, and sorting data.
6. Explain the architecture and working of Apache Hive.
7. Differentiate between Managed Tables and External Tables.
8. Create partitions and buckets in Hive.
9. Import, modify, query, and delete data in Hive.
10. Write simple MapReduce scripts for processing large datasets.

Importance of Studying this Unit

Writing MapReduce programs in Java is complex and time-consuming. Apache Pig and Apache Hive simplify Big Data processing by providing high-level scripting and SQL-like query languages. These tools improve productivity, reduce coding effort, and are widely used in data engineering, business intelligence, and analytics.

Key Concepts: Apache Pig, Pig Latin, Local Mode, MapReduce Mode, Data Types, Schemas, Functions, Data Processing Operators, Apache Hive, HiveQL, Managed Table, External Table, Partitioning, Bucketing, Data Import, Query Processing, Aggregation, Sorting, MapReduce Script.

PIG

- 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.

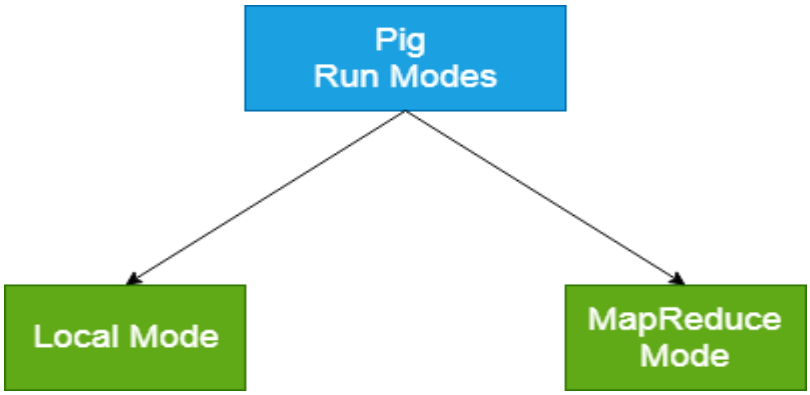
Pig vs MapReduce

Apache Pig	MapReduce
It is a scripting language.	It is a compiled programming language.
Abstraction is at higher level.	Abstraction is at lower level.
It have less line of code as compared to MapReduce.	Lines of code is more.
Less effort is needed for Apache Pig.	More development efforts are required for MapReduce.
Code efficiency is less as compared to MapReduce.	As compared to Pig efficiency of code is higher.
Pig provides built in functions for ordering, sorting and union.	Hard to perform data operations.
It allows nested data types like map, tuple and bag	It does not allow nested data types

Pig Execution Types

Apache Pig executes in two modes:

1. **Local Mode and**
2. **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:

$ pig-x local

Map Reduce 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:

$ pig -x mapreduce

Pig Example

A = Load 'rp/WeatherDataset.txt' as (line:chararray);

B = foreach A generate
SUBSTRING(line,15,19),SUBSTRING(line,88,92),SUBSTRING(line,92,93);

store B into 'rp/revised5';

E = load 'rp/revised5/part-m-00000' using PigStorage('\t') as (year:int,temp:int,quality:int);

```
filtered_records = FILTER E by temp!=9999 AND quality IN (0,1,4,5,9);

grouped_records = GROUP filtered_records by year;

max_temp = FOREACH grouped_records GENERATE group,MAX(filtered_records.temp);

dump max_temp;
```

Pig Latin

- Pig Latin is the Language used in Pig
- Pig Latin Structure includes

  Comments

  Statements

Comments

- Proper use of commenting can make code maintenance much easier, as well as helps finding bugs faster.
- Pig Latin has two types of comment operators:

  SQL-style single-line comments (--) and

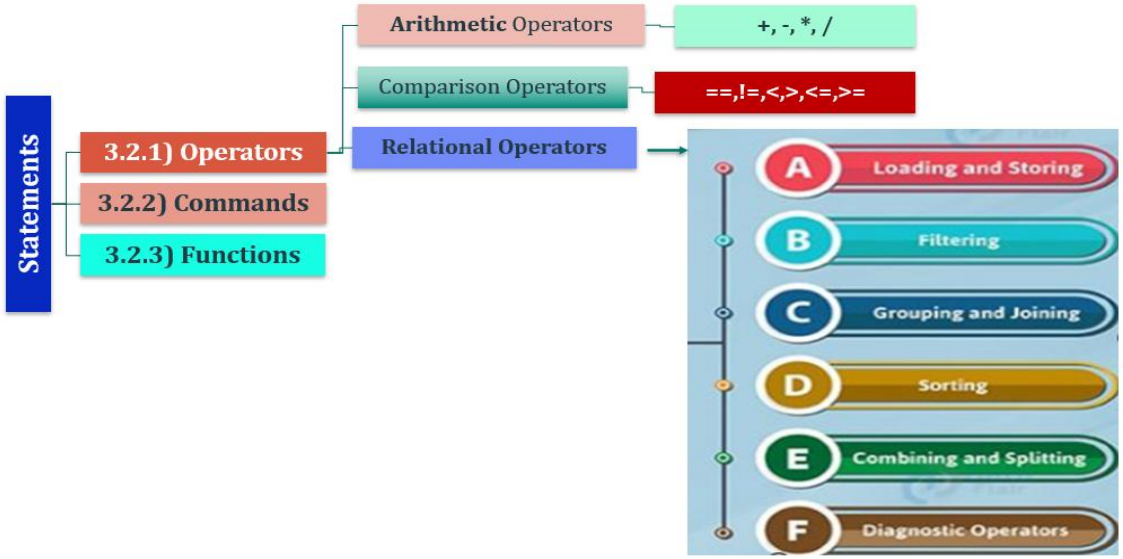
  Java-style multiline comments (/* */).

Statements

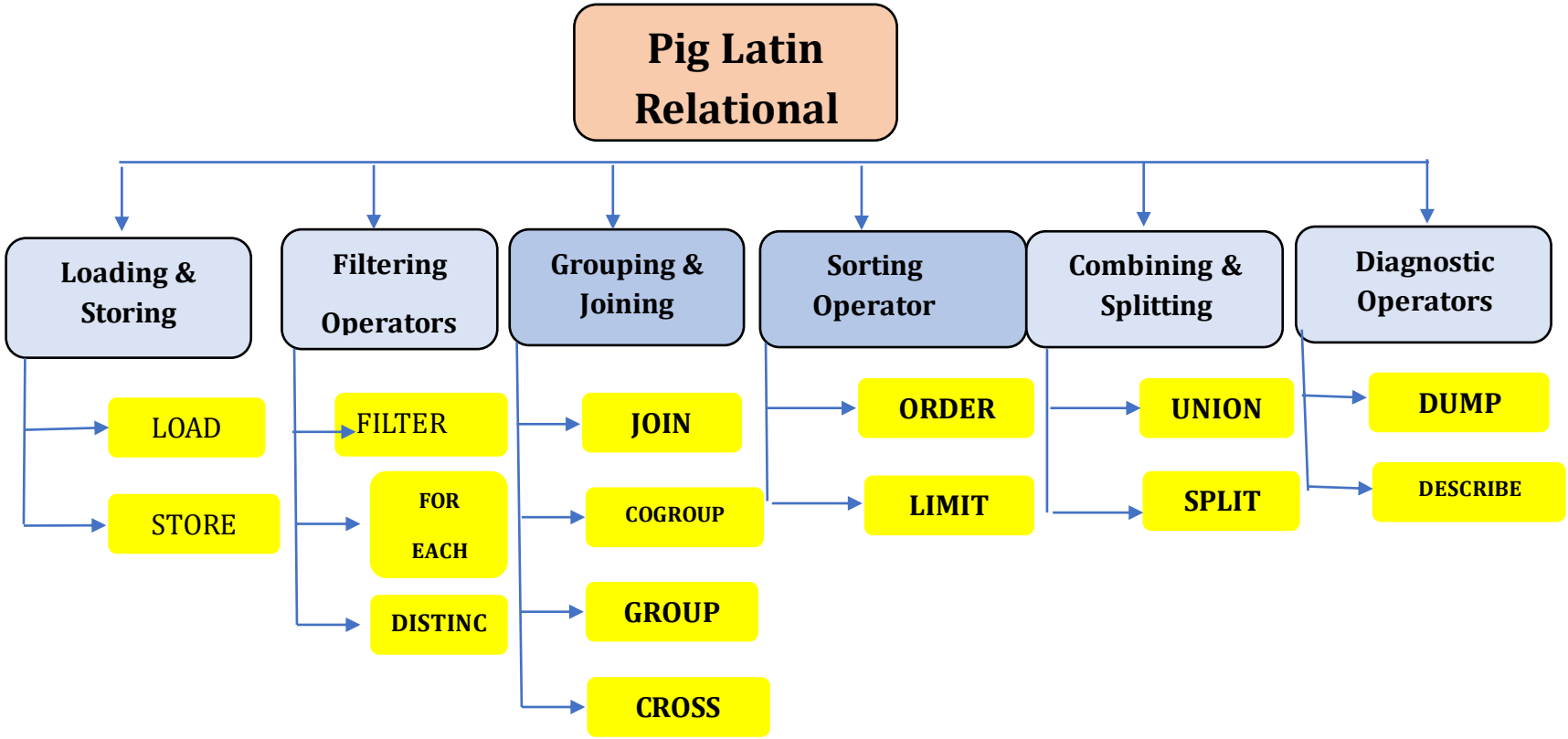
The statements are the basic constructs while processing data using Pig Latin.

- The statements can work with relations including expressions and schemas.
- However, every statement terminate with a semicolon (;).
- We will perform different operations using Pig Latin operators.
- Pig Latin statements inputs a relation and produces some other relation as output.
- As a Pig Latin Program is executed ,each statement is parsed in turn If there are syntax errors or other problems, such as undefined aliases, the interpreter will halt and display an error message.
- The Interpreter builds a logical plan for every relational operation, which forms the core of a pig Latin program.
- The Logical Plan for the statement is added to the logical plan for the program to far, and then the interpreter moves on to the next statement
- Its important to note that no data processing takes place while the logical plan of the program is being constructed
- When the Pig Latin interpreter sees the first line containing the load statement. It conforms that it is syntactically and semantically correct and adds it to the logical plan, but it does not load the data from the file .Indeed, where would it load it? Into memory?even if it did fit into memory, what would it do with data
- Similarly , pig validates GROUP and FOREACH statements and adds them to the logical plan without executing them

- The Trigger for Pig to start execution is the Dump statement
- At that point, the logical plan is compiled into a physical plan and executed.
- Pig Latin Statements Includes
    - **Operators**
    - **Commands**
    - **Functions**



Statements & operators



Loading and Storing Operator

Load Operator

Is used to load data into Apache Pig from HDFS

Syntax

Rel_Name = LOAD 'input file path' using PigStorage('\t') as schema

where

Rel_Name – Name of the Relation where to store the data

Input file path – path of the HDFS file

Scheme – Resultant Relation schema

Eg:

A= LOAD '/data/sample.txt' as PigStorage('\t') as (sno:int,sname:chararray,branch:chararray)

- The Load statement consists of 2 parts by the "=" operator
- On the left hand side, need to mention the relation name where we want to store the data and
- On the right hand side, we have to define how to store the data (schema)

$ cat > sample.txt

100 smith MCA

101 Jones M.Tech

102 King MCA

103 Ivan MCA

104 Korth M.Tech

Ctrl +Z

$ hadoop fs –mkdir /data → Create a Directory called data in HDFS

$ hadoop fs –put sample.txt /data → Push sample.txt from local file system to HDFS data directory

$ hadoop fs -ls /data → check sample.txt exists in data directory

$ pig → Move to Pig

grunt> student = LOAD '/data/sample.txt' using PigStorage('\t') as

(sno:int, sname:chararray,branch:chararray);

grunt>dump student

(100 smith MCA)

(101 Jones M.Tech)

(102 King MCA)

(103 Ivan MCA)

(104 Korth M.Tech)

Load Operator

Purpose	Syntax	Example	Output
To Load Data from HDFS to Pig latin	Rel_Name = LOAD 'input file path' using PigStorage('\t') as schema where ➢ Rel_Name – Name of the Relation where to store the data ➢ Input file path – path of the HDFS file ➢ Scheme – Resultant Relation schema	grunt> student = LOAD '/data/sample.txt' using PigStorage('\t') as (sno:int, sname:chararray, branch :chararray); grunt>dump student	(100 smith MCA) (101 Jones M.Tech) (102 King MCA) (103 Ivan MCA) (104 Korth M.Tech)

Store Operator

Purpose	Syntax	Example	Output

Is used to store data from Apache Pig to HDFS	STORE Rel_Name INTO 'HDFS Directory';
	where
	Rel_Name – Name of the Relation which needs to be stored from pig to HDFSc
	HDFS Directory – HDFS Path where to store
		$ hadoop fs –mkdir /Results
		$ pig
		grunt>STORE student into '/Results';
			$ hadoop fs –cat /Results/part-m-00000
			100 smith MCA
			101 Jones M.Tech
			102 King MCA
			103 Ivan MCA
			104 Korth M.Tech

Filter Operator

Purpose	Syntax	Example	Output
Is used FILTER Rows from the Relation Is used to get specific rows of the Relation It is equivalent to WHERE clause in SQL	Res_Relation = FILTER Rel_Name BY Condition where Rel_Name – Name of the Relation on which filter need to be performed Condition– Filter condition	Grunt>MCA_Stu = FILTER student BY branch == 'MCA'; Grunt>M.Tech_Stu = FILTER student BY branch == 'MCA';	Grunt>dump MCA_Stu; (100 smith MCA) (102 King MCA) (103 Ivan MCA) Grunt>dump M.Tech_Stu ; 101 Jones M.Tech 104 Korth M.Tech

ForEach .. Generate Operator

Purpose	Syntax	Example	Output
Is used to get specific columns It is equivalent to SELECT clause in SQL	Res_Relation = FOREACH Rel_Name GENERATE COL1,COL2,…COLN Where Rel_Name – Name of the Relation from which specific columns to be retreived COL1,COL2,..COLN are the specific columns to be rerived	Grunt> no_branch = FOREACH student GENERATE sno,branch; If schema is not defined then Grunt>no_branch = FOREACH student GENERATE $1,S3;	Grunt>dump no_banch; (100 MCA) (101 M.Tech) (102 MCA) (103 MCA) (104 M.Tech)

GROUPING & JOINING Operators

JOIN OPERATOR

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Purpose	Syntax	Example	Output
Performs an inner join of two or more relations based on common field values.	Res_Relation = JOIN Rel1 by Key, Rel2 by key where Rel1 is the First Relation Rel2 is the Second Relation Key is the common field	grunt> emp_dept = JOIN emp BY deptno ,dept BY deptno	Grunt>dump emp_dept; eno ename deptno deptno dname 100 Smith 10 10 HR 101 Jones 20 20 SYSTEM 102 King 10 10 HR 103 Ivan 20 20 SYSTEM 104 Korth 30 30 MARKETING

COGROUP OPERATOR

- Join always give a flat structure
- The COGROUP statement is similar to join but instead creates a nested set of output tuples.
- COGROUP generates a tuple for each unique grouping key
  - ✓ The first field of each tuple is the key
  - ✓ And the remaining fields are bags of tuples from the relations with a matching key
  - ✓ The first bag contains the matching tuples from relation A with the same key.
  - ✓ The second bag contains the matching tuples from relation B with the same key

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Purpose	Syntax	Example	Output
Same as Join instead JOIN returns flat structure whereas COGROUP returns nested tuples	Res_Relation = COGROUP Rel1 by Key, Rel2 by key where Rel1 is the First Relation Rel2 is the Second Relation Key is the common field	grunt> emp_dept= COGROUP emp BY deptno ,dept BY deptno	Grunt>dump emp_dept; (10, {(100,Smith,10),(102,King,10)},{(10,HR)}) (20, {(101,Jones,20),(103,Ivan,20)},{(20,SYSTEM)}) (30, {(104,Korth,30)},{(30,MARKETING)})

GROUP OPERATOR

Purpose	Syntax	Example	Output
Join and Cogroup works on one or more relations whereas GROUP operator is used to group data in a single relation	Res_Relation = GROUP Rel by Field where Rel is the Relation Field is column or field on which to group the data	grunt> deptwise = GROUP emp by deptno	Grunt>dump deptwise; (10, {(100,Smith,10),(102,King,10)}) (20, {(101,Jones,20),(103,Ivan,20)}) (30, {(104,Korth,30)})

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

CROSS OPERATOR

Purpose	Syntax	Example	Output
Returns Cross Product of 2 or more Relations	Res_Relation = CROSS Rel1,Rel2 where Rel1 and Rel2 are Relations on which cross product need to be applied	grunt> cp = CROSS emp,dept;	**Grunt>dump cp;** 100 Smith 10 10 HR 10 100 Smith 10 20 SYSTEM 20 101 Jones 20 10 HR 10 101 Jones 20 20 SYSTEM 20 102 King 10 10 HR 10 102 King 10 20 SYSTEM 20

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM

Sorting and Limiting Data

ORDER OPERATOR

Purpose	Syntax	Example	Output
Returns data in Sorted order	Res_Relation = ORDER Rel1 by FieldName where Rel1 is the Relation FieldName is the column on which data to be sorted	grunt> sorted_names = ORDER emp BY ename;	Grunt>dump sorted_names; (103 Ivan 20) (101 Jones 20) (102 King 10) (104 Korth 30) (100 Smith 10)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

LIMIT OPERATOR

Purpose	Syntax	Example	Output
is used to get a limited number of tuples from a relation.	Res_Relation = LIMIT Rel1 Number where Rel1 is the Relation Number is Number of tuples to be displayed	grunt>Limited_tuples = LIMIT emp 2;	Grunt>dump sorted_names; (100 Smith 10) (101 Jones 20)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Combining and Splitting Data

UNION OPERATOR

Purpose	Syntax	Example	Output
is used to combine 2 or more relations into one	Res_Relation = UNION Rel1,Rel2 where Rel1 is the First Relation Rel2 is the Second Relation	grunt>combined_rels = UNION emp,dept;	Grunt>dump combined_rels; (100 Smith 10) (101 Jones 20) (102 King 10) (103 Ivan 20) (104 Korth 30) (10 HR) (20 SYSTEM) (30 MARKETING)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

104	**Korth 30**	

SPLIT OPERATOR

Purpose	Syntax	Example	Output
is used to Split single Relation into 2 or more Relations	Split Rel1 into Rel2 if condition1 , Rel3 if condition2 where Rel1, Rel2, Rel3 are Relations. Condition1 and condition 2 are conditions on which the Rel1 to be split	grunt>split emp into deptno10 if deptno ==10, deptno20 if deptno ==20;	grunt>dump deptno10; (100 Smith 10) (102 King 10) Grunt>dump deptno20; (101 Jones 20) (103 Ivan 20)

grunt>dump emp		
eno	**ename**	**deptno**
100	**Smith**	**10**
101	**Jones**	**20**
102	**King**	**10**

grunt>dump dept	
deptno	**dname**
10	**HR**
20	**SYSTEM**
30	**MARKETING**

103	**Ivan** **20**	
104	**Korth** **30**	

Diagnostic Operators

DUMP OPERATOR

Purpose	Syntax	Example	Output
is used to Display Results on the Screen	DUMP Relation_Name; where Relation_Name is the name of the Relation whose content to be printed	grunt>DUMP emp;	grunt>dump emp; (100 Smith 10) (101 Jones 20) (102 King 10) (103 Ivan 20) (104 Korth 30)

grunt>dump emp		
eno	**ename**	**deptno**
100	**Smith**	**10**
101	**Jones**	**20**

grunt>dump dept	
deptno	**dname**
10	**HR**
20	**SYSTEM**

102	**King**	**10**	**30**	**MARKETING**
103	**Ivan**	**20**		
104	**Korth**	**30**		

DESCRIBE OPERATOR

Purpose	Syntax	Example	Output
describes the schema of a relation.	DESCRIBE Relation_Name; where Relation_Name is the name of the Relation	grunt>DESCRIBE emp;	grunt>DESCRIBE emp; emp: { eno:int,ename:chararray,deptno:int}

grunt>dump emp			**grunt>dump dept**	
eno	**ename**	**deptno**	**deptno**	**dname**
100	**Smith**	**10**	**10**	**HR**
101	**Jones**	**20**	**20**	**SYSTEM**
102	**King**	**10**	**30**	**MARKETING**
103	**Ivan**	**20**		
104	**Korth**	**30**		

PIG 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** : 'tutorials point'
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]

SCHEMAS

- Schemas enable to assign names and types to fields .
- Schemas are optional but we encourage you to use them whenever possible; type declarations result in better parse-time error checking and more efficient code execution.
- Schema can be defined for both names and types of the fields

- Schema can be defined only for field names, in this case, the field type defaults to bytearray. Here fields can be referred through names
- Schema may not be defined ; in this case, the field is un-named and the field type defaults to bytearray. Here the fields can be referred through positional notation like $1,$2,etc..

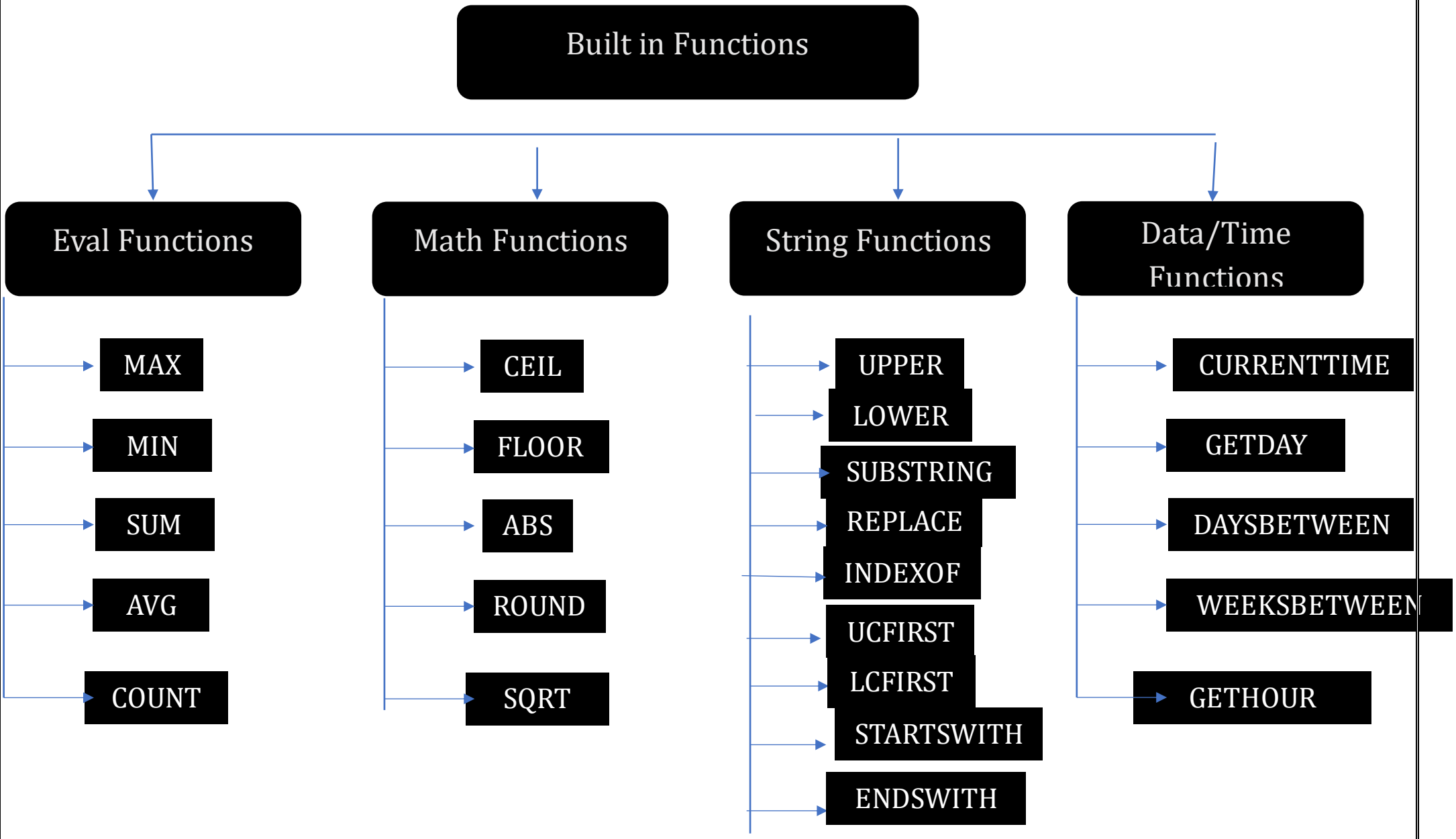
Unknown Schema Handling

- When you JOIN/COGROUP/CROSS multiple relations, if any relation has an unknown schema (or no defined schema, also referred to as a null schema), the schema for the resulting relation is null.
- If you UNION two relations with incompatible schema, the schema for resulting relation is null.

Schema Merging

- In Pig, no need to declare the schema for every new relation in the data flow
- pig can figure out the resulting schemas for the output of a relational operation by considering the schema of the input relation
- Filter , limit are the operators who resultant relation schema is same as input schema

Pig Latin Built in Functions



S.NO.	FUNCTION & DESCRIPTION
1	AVG() -To compute the average of the numerical values within a bag.
2	COUNT() -To get the number of elements in a bag, while counting the number of tuples in a bag.
3	MAX() -To calculate the highest value for a column (numeric values or chararrays) in a single-column bag.
4	MIN() -To get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag.
5	SUM() -To get the total of the numeric values of a column in a single-column bag.

$ cat > Student_details

Smith wp 67

Jones Java 78

Ivan BDA 85

Ivan java 79

Jones BDA 69

Smith Java 89

Ctrl+z

$ hadoop fs –mkdir /data

$ hadoop fs –put Student_details /data

$ pig

grunt> Student = LOAD '/data/Student_details' using PigStorage('\t') as sname:chararray,subject :chararray,Marks:int);

grunt>dump Student;

S.NO.	FUNCTION & DESCRIPTION
1	AVG() -To compute the average of the numerical values within a bag.
2	COUNT() -To get the number of elements in a bag, while counting the number of tuples in a bag.
3	MAX() -To calculate the highest value for a column (numeric values or chararrays) in a single-column bag.
4	MIN() -To get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag.
5	SUM() -To get the total of the numeric values of a column in a single-column bag.

$ cat > Student_details

Smith wp 67

Jones Java 78

Ivan BDA 85

Ivan java 79

Jones BDA 69

Smith Java 89

Ctrl+z

$ hadoop fs –mkdir /data

$ hadoop fs –put Student_details /data

$ pig

grunt> Student = LOAD '/data/Student_details' using PigStorage('\t') as

sname:chararray,subject :chararray,Marks:int);

grunt>dump Student;

Sname	subject	Marks
Smith	wp	67
Jones	Java	78
Ivan	BDA	85
Ivan	java	79
Jones	BDA	69

1) Find Highest Marks Obtained in each Subject

```
grunt> subjectwise = group student by subject;

grunt>dump subjectwise;

    (wp,{(smith,wp,67)})

    (Java,{(Jones,Java,78),(Ivan,Java,79)})

    (BDA,{(Ivan,BDA,85),(Jones,BDA,69)})

grunt>highest = foreach subjectwise generate group,MAX(student.Marks);

grunt>dump highest;

   (wp,67)

   (Java,79)

   (BDA,85)
```

2) Find Lowest Marks Obtained in each Subject

```
grunt> subjectwise = group student by subject;

grunt>dump subjectwise;

    (wp,{(smith,wp,67)})

    (Java,{(Jones,Java,78),(Ivan,Java,79)})

    (BDA,{(Ivan,BDA,85),(Jones,BDA,69)})

grunt>lowest = foreach subjectwise generate group,MIN(student.Marks);

grunt>dump lowest;

   (wp,67)  (Java,78)     (BDA,69)
```

3) Find No.of Students in each subject

```
grunt> subjectwise = group student by subject;

grunt>dump subjectwise;

    (wp,{(smith,wp,67)})

    (Java,{(Jones,Java,78),(Ivan,Java,79)})

    (BDA,{(Ivan,BDA,85),(Jones,BDA,69)})

grunt>NoOfStudents = foreach subjectwise generate groupCOUNT(student.Marks);

grunt>dump NoOfStudents;

   (wp,1)  (Java,2) (BDA,2)
```

4) Find Total Marks Obtained by each Student

```
grunt> studentwise = GROUP student by sname;
grunt>dump studentwise;
  (smith,{(smith,wp,67)})
  (Jones,{(Jones,Java,78),(Jones,BDA,69)})
  (Ivan,{(Ivan,Java,79),(Ivan,BDA,85)})
grunt>tot = foreach studentwise generate group,SUM(student.Marks);
grunt>dump tot;
   (smith,67) (Jones,147)(Ivan,164)
```

5) Find Percentage Obtained by each Student

```
grunt> studentwise = GROUP student by sname;
grunt>dump studentwise;
  (smith,{(smith,wp,67)})
  (Jones,{(Jones,Java,78),(Jones,BDA,69)})
  (Ivan,{(Ivan,Java,79),(Ivan,BDA,85)})
grunt>percentage = foreach studentwise generate group,AVG(student.Marks);
grunt>dump percentage;
   (smith,67) (Jones,73.5)(Ivan,82)
```

6) Find Number of subjects taken by each student

```
grunt> studentwise = GROUP student by sname;
grunt>dump studentwise;
  (smith,{(smith,wp,67)})
  (Jones,{(Jones,Java,78),(Jones,BDA,69)})
  (Ivan,{(Ivan,Java,79),(Ivan,BDA,85)})
grunt>NoOfSubjects = foreach studentwise generate group,COUNT(student.Marks);
grunt>dump NoOfSubjects;;
   (smith,1)
   (Jones,2)
   (Ivan,2)
```

S.NO.	FUNCTIONS & DESCRIPTION
1	ABS(expression) -To get the absolute value of an expression.
2	CBRT(expression)-This function is used to get the cube root of an expression.
3	CEIL(expression) -This function is used to get the value of an expression rounded up to the nearest integer.
4	FLOOR(expression) -To get the value of an expression rounded down to the nearest integer.
5	ROUND(expression) -To get the value of an expression rounded to an integer (if the result type is float) or rounded to a long (if the result type is double).
6	SQRT(expression) -To get the positive square root of an expression.

```
$ cat > Stu_Grades
Smith 7.4
Jones 8.2
Ivan  9.6
Korth 6.9
King  5.4
Ctrl+z
```

```
$ hadoop fs –put Stu_Grades  /data
$ pig
grunt> grades = load '/data/Stu_Grades' using PigStorage('\t')
as sname:chararray,cgpa:float;
grunt>dump grades;
sname       cgpa
Smith 7.4
Jones 8.2
Ivan  9.6
Korth 6.9
King  5.4
```

```
grunt> Math = ForEach grades Generate cgpa,
CEIL(cgpa),FLOOR(cgpa),ABS(cgpa),ROUND(cgpa);
grunt>dump Math;
    (7.4,  8,    7,    7.4,   7.4)
    (8.2,  9,    8,    8.2,   8.2)
    (9.6,  10,   9,    9.6,   10)
    (6.9,  7,    6,    6.9,    7)
    (5.4,  6,    5,    5.4,   5.4)
```

S.NO.	Functions & Description
1	ENDSWITH(string, testAgainst) To verify whether a given string ends with a particular substring.
2	STARTSWITH(string, substring) Accepts two string parameters and verifies whether the first string starts with the second.
3	SUBSTRING(string, startIndex, stopIndex) Returns a substring from a given string.
4	EqualsIgnoreCase(string1, string2) To compare two stings ignoring the case.
5	INDEXOF(string, 'character', startIndex) Returns the first occurrence of a character in a string, searching forward from a start index.
6	LAST_INDEX_OF(expression) Returns the index of the last occurrence of a character in a string, searching backward from a start index.
7	LCFIRST(expression) Converts the first character in a string to lower case.
8	UCFIRST(expression) Returns a string with the first character converted to upper case.
9	UPPER(expression) UPPER(expression) Returns a string converted to upper case.
10	LOWER(expression) Converts all characters in a string to lower case.
11	REPLACE(string, 'oldChar', 'newChar'); To replace existing characters in a string with new characters.

grunt>dump grades;

sname cgpa

Smith 7.4

Jones 8.2

Ivan 9.6

Korth 6.9

grunt> str = foreach grades generate sname, LOWER(sname),UPPER(sname), SUBSTRING(sname,1,3);

grunt> dump str;

(Smith smith SMITH mi)

(Jones jones JONES on)

(Ivan ivan IVAN va)

(Korth korth KORTH or)

grunt>str1 = foreach grades generate sname, ENDSWITH(sname,'h'), STARTSWITH(sname,'k'),LCFIRST(sname),UCFIRST(sname);

grunt>dump str1;

(Smith TRUE FALSE smith Smith)

(Jones FALSE FALSE jones Jones)

(Ivan FALSE FALSE ivan Ivan)

(Korth TRUE TRUE korth Korth)

grunt>str2 = foreach grades generate sname, REPLACE(sname,'i',z'), INDEXOF(sname,'t');

grunt>dump str2;

(Smith Smzth 3)

(Jones Jones -1)

(Ivan zvan -1)

(Korth Korth 3)

Data/Time Functions

S.NO.	Functions & Description
1	CurrentTime() :returns the date-time object of the current time.
2	GetDay(datetime) :Returns the day of a month from the date-time object.
3	GetHour(datetime) :Returns the hour of a day from the date-time object.
4	GetMonth(datetime) Returns the month of a year from the date-time object.
5	GetSecond(datetime) :Returns the second of a minute from the date-time object.
6	GetWeek(datetime) :Returns the week of a year from the date-time object.
7	DaysBetween(datetime1, datetime2) :Returns the number of days between the two date-time objects.
8	HoursBetween(datetime1, datetime2) :Returns the number of hours between two date-time objects.
9	MonthsBetween(datetime1, datetime2) :Returns the number of months between two date-time objects.
10	SecondsBetween(datetime1, datetime2) :Returns the number of seconds between two date-time objects.
11	WeeksBetween(datetime1, datetime2) :Returns the number of weeks between two date-time objects.
12	YearsBetween(datetime1, datetime2): Returns the number of years between two date-time objects.

HIVE

Apache Hive is an open-source data warehouse software built on top of the Hadoop ecosystem. It provides a SQL-like query language called Hive Query Language (HiveQL) that enables users to store, manage, query, and analyze large datasets stored in the Hadoop Distributed File System (HDFS). Hive converts HiveQL queries into MapReduce, Apache Tez, or Apache Spark jobs for distributed processing.

Hive was originally developed by Facebook to simplify the processing of massive datasets and later became an Apache Software Foundation project.

Need for Hive

Writing MapReduce programs in Java is complex and time-consuming. Hive provides an SQL-like interface that allows users familiar with SQL to perform data analysis without writing Java code.

Hive is mainly used for:

- Data warehousing
- Business intelligence
- Data summarization
- Batch processing
- Report generation
- Large-scale data analysis

HiveQL Data Definition Language (DDL)

HiveQL Data Definition Language (DDL) is a set of commands used to define and manage the structure of databases and tables in Apache Hive. DDL commands are used to create, modify, describe, and delete databases, tables, views, partitions, and other schema objects. These commands define how data is organized and stored in the Hadoop Distributed File System (HDFS).

Table Creation in Hive

Hive supports two types of tables:

1. Internal Table (Managed Table)
2. External Table

Internal Table

An Internal Table is managed completely by Hive. When the table is dropped, both the table schema (metadata) and the data stored in HDFS are deleted.

Syntex:

```
create table emp (eid int, ename string, esal int, eloc string)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;
```

Loading Data:

1. load data local inpath 'sample1.txt' into table emp;
2. load data inpath '/yuvaraj/sample2.txt' into table emp;
3. insert overwrite table emp1 select * from emp;

External Table

Only metadata is deleted when the table is dropped; data remains in HDFS.

Syntex:

```
create external table emp_ex (eid int, ename string, esal int, eloc string)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;
```

Loading Data:

1. load data local inpath 'sample1.txt' into table emp_ex;
2. load data inpath '/yuvaraj/sample2.txt' into table emp_ex;
3. insert overwrite table emp_ex select * from emp1;

Partitioning

- Hive table partition is a way to split a large table into smaller logical tables based on one or more partition keys.

- These smaller logical tables are not visible to users and users still access the data from just one table.
- When you load the data into the partition table, Hive internally splits the records based on the partition key and stores each partition data into a sub-directory of tables directory on HDFS.
- The name of the directory would be partition key and it's value

Partition Table Advantages

1) Fast accessed to the data
2) Provides the ability to perform an operation on a smaller dataset.

Syntex:

```
create table emp_part (eid int, ename string, esal int)
partitioned by (eloc string)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;
```

Static Partition:

```
insert overwrite table emp_part partition(eloc='hyd') select eid,ename,esal from emp1 where eloc='hyd';
insert overwrite table emp_part partition(eloc='bng') select eid,ename,esal from emp1 where eloc='bng';
```

dynamic partition:

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table emp_part partition(eloc) select eid,ename,esal,eloc from emp1;
```

Bucketing

- Hive Bucketing is a way to split the table into a managed number of clusters with or without partitions.

- Hive Buckets(Clustering) is a technique to split the data into more manageable files, (By specifying the number of buckets to create).
- The value of the bucketing column will be hashed by a user-defined number into buckets.
- Each bucket is stored as a file within the table's directory or the partitions directories on HDFS.

Hive bucketing commonly created in two scenarios.

- Create a bucket on top of the Partitioned table to further divide the table for better query performance.
- Create Bucketing on the table where you cannot choose the partition column due to (too many distinct values on columns).

Create Hive Buckets

- To create a Hive table with bucketing, use CLUSTERED BY clause with the column name you wanted to bucket and the count of the buckets.

Syntex:

```
create table emp_cluster(eid int, ename string, esal int, eloc string)
clustered by (eid) into 4 buckets
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;
```

Load Data into Bucket

```
insert overwrite table emp_cluster select * from emp1;
```

List Buckets:

```
hdfs dfs -ls /user/hive/warehouse/emp_cluster
```

Importing data

Hive provides multiple ways to add data to the tables. In hive with DML statements, we **can add data to the Hive table in 2 different ways**.

- **Using INSERT Command**
- **Load Data Statement**

Using INSERT Command

Syntax:

INSERT INTO TABLE <table_name> VALUES (<add values as per column entity>);

Example:

- To insert data into the table let's create a table with the name *student* (By default hive uses its *default* database to store hive tables).

Command:

```
hive> CREATE TABLE IF NOT EXISTS student(
    > Student_Name STRING,
    > Student_Rollno INT,
    > Student_Marks FLOAT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 2.086 seconds
hive> show tables in default;
OK
student
Time taken: 0.268 seconds, Fetched: 1 row(s)
hive>
```

INSERT Query:

INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat', 2 , '96'),('Dhruv',3,'90');

```
hive> INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat', 2 , '96'),
('Dhruv',3,'90');
Query ID = dikshant_20201106121659_f5dfa694-f552-4b7a-a64b-4f3804213ab8
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
```

We can check the data of the *student* table with the help of the below command.

SELECT * FROM student;

```
hive> SELECT * FROM student;
OK
Dikshant        1       95.0
Akshat  2       96.0
Dhruv   3       90.0
Time taken: 0.162 seconds, Fetched: 3 row(s)
hive>
```

Load Data Statement

- Hive provides us the functionality to load pre-created table entities either from our local file system or from HDFS. The *LOAD DATA* statement is used to load data into the hive table.

Syntax:

LOAD DATA [LOCAL] INPATH '<The table data location>' [OVERWRITE] INTO TABLE <table_name>;

- The LOCAL Switch specifies that the data we are loading is available in our Local File System. If the LOCAL switch is not used, the hive will consider the location as an HDFS path location.
- The OVERWRITE switch allows us to overwrite the table data.
- LOAD DATA to the student hive table with the help of the below command.

LOAD DATA LOCAL INPATH '/home/dikshant/Documents/data.csv' INTO TABLE student;

```
hive> LOAD DATA LOCAL INPATH '/home/dikshant/Documents/data.csv' INTO TABLE stud
ent;
Loading data to table default.student
OK
Time taken: 2.617 seconds
hive>
```

alter table:

ALTER command is used to Alter the Schema of the table

ALTER command is used to

1) Add the Column
2) change the Column
3) Replace the Column Name
4) Replace the Table Name

Add the Column to the Table

Syntax

ALTER TABLE <table_name> ADD COLUMNS (<col-name> <data-type> COMMENT ' ', <col-name> <data-type> COMMENT ' ', …..)

Example

ALTER TABLE customer ADD COLUMNS (contact BIGINT COMMENT 'Store the customer contact number');

```
hive> ALTER TABLE customer ADD COLUMNS ( contact BIGINT COMMENT 'Store the custo
mer contact number');
OK
Time taken: 0.093 seconds
```

CHANAGE Column

- CHANGE in ALTER TABLE is used to change the name or data type of an existing column or attribute.

Syntax:

ALTER TABLE <table_name> CHANGE <column_name> <new_column_name> <new_data_type>;

- Let's change the demo_name attribute to customer_name.

ALTER TABLE customer CHANGE demo_name customer_name STRING;

```
hive> ALTER TABLE customer CHANGE demo_name customer_name STRING;
OK
Time taken: 0.131 seconds
hive> describe customer;
OK
customer_name           string
contact                 bigint                  Store the customer contact numbe
r
Time taken: 0.05 seconds, Fetched: 2 row(s)
hive>
```

REPLACE Column

- The REPLACE with ALTER TABLE is used to remove all the existing columns from the table in Hive.
- The attributes or columns which are added in the ALTER TABLE REPLACE statement will be replaced with the older columns.

Syntax:

ALTER TABLE <table_name> REPLACE COLUMNS (<attribute_name> <data_type>, <attribute_name> <data_type>, . . .);

- For example in our customer table, we have 2 attributes customer_name and contact. If we want to remove the contact attribute the query should be like as shown below.

Command:

ALTER TABLE customer REPLACE COLUMNS (customer_name STRING);

```
hive> ALTER TABLE customer REPLACE COLUMNS (
    > customer_name STRING
    > );
OK
Time taken: 0.143 seconds
hive> describe customer;
OK
customer_name           string
Time taken: 0.068 seconds, Fetched: 1 row(s)
hive>
```

Renaming Table Name

- ALTER TABLE with RENAME is used to change the name of an already existing table in the hive.

Syntax:

ALTER TABLE <current_table_name> RENAME TO <new_table_name>;

Command:

Let's rename our table name from the **demo** to the **customer**.

ALTER TABLE demo RENAME TO customer;

Dropping Tables

Drop Table Statement drops the table from hive metastore.

It removes the data and their metadata

Syntax

DROP TABLE [if exists] Table_Name

Example

DROP TABLE employee;

Querying Data-Sorting and Aggregating

In Hive, querying data is performed by a SELECT statement. A SELECT statement has 6 key components

1. SELECT columnnames
2. FROM table_name
3. GROUP BY column_names
4. WHERE condition
5. HAVING condition
6. ORDER BY column_name

Aggregating Functions

1) AVG
2) SUM
3) COUNT
4) MIN
5) MAX

MapReduce Scripts.

MapReduce Scripts in Hive allow you to process data using custom scripts or programs (such as Python, Bash, Perl, or Java) instead of writing Java MapReduce code. Hive sends data to the script through standard input (STDIN) and receives the processed output through standard output (STDOUT).

This feature is useful when complex data transformations cannot be easily implemented using HiveQL.

Case Study: Employee Salary Analysis Using Hive Window Functions

Title

Analyzing Employee Performance and Salary Rankings Using Apache Hive

Problem Statement

ABC Technologies has more than **10,000 employees** working in different departments such as HR, IT, Sales, and Finance. The HR department wants to analyze employee salaries to:

- Identify the highest-paid employees in each department.
- Assign ranks based on salary.
- Handle employees having the same salary correctly.
- Select the top three employees from each department for annual bonuses.

Since the company stores millions of employee records in Hadoop, it uses **Apache Hive** to perform SQL-like analysis efficiently.

Unit Highlights

- Apache Pig simplifies MapReduce programming through Pig Latin.
- Pig supports Local Mode and MapReduce Mode for execution.
- Pig provides powerful data processing operators such as FILTER, GROUP, JOIN, ORDER, and DISTINCT.
- Apache Hive is a data warehouse tool that uses HiveQL, an SQL-like language.
- Hive supports Managed Tables and External Tables for flexible data management.
- Partitioning and Bucketing improve query performance on large datasets.
- Hive supports data import, alteration, querying, sorting, and aggregation operations.
- Pig and Hive are widely used for ETL, data warehousing, and business intelligence in Big Data environments.

TEXT BOOKS:

1. Big Data Fundamentals: Concepts, Drivers & Techniques", 1/e, 2016, Thomas Erl, Wajid Khattak, Paul Buhler, Prentice Hall.
2. "Hadoop:The Definitive Guide**,"** 3/e, 2012, Tom White, O'REILLY Publications.
3. "Learning Spark"

REFERENCE BOOKS:

1. Taming the Big Data Tidal Wave: Finding Opportunities in Huge Data Streams with Advanced Analytics, 2012, Bill Franks, John Wiley & Sons..
2. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, 2012, Paul C. Zikopoulos, Chris Eaton, Dirk deRoos, Thomas Deutsch, George Lapis, McGraw-Hill.
3. Intelligent Data Analysis, 2007, Michael Berthhold, David J.Hand, Springer.
4. Understanding Big Data: Analytics for Enterprise Class Hadoop and streaming data, 2011, Paul Zikopoulos, Chris Eaton, Paul Zikopoulos, McGraw hill.
5. Big Data for Dummies, 2012, Hurwitz, Alan Nugent, Dr. Fern Halper, Marcia Kaufman,John Wiley & Sons

S.No	Questions
UNIT III – Working with Pig and Hive	
PART-A (Two Marks Questions)	
1.	What is Pig
2.	List out the Pig Execution Types
3.	Mention the Language used in pig
4.	List out the Pig Data Types
5.	List out two eval functions of pig
6.	Purpose of LOAD operator
7.	What is hive
8.	Mention 2 types of Hive Tables
9.	Write Two ways to import data into hive tables
10.	Write about Alter statement to add columns
11.	Which clause is used to partition Tables
12.	Which clause is used to sort the table data
13.	Which clause is used to do bucketization
14.	Explain the main advantage of partitions and buckets
15.	Write the syntax to drop a table in hive
PART-B (Ten Marks Questions)	
1.	Write a Pig Script to analyze weather data set
2.	Discuss the Eval and string functions of Pig
3.	Write about Load and Store operators
4.	Explain Math and String functions in pig
5.	Discuss Pig Data Processing Operators with example
6.	Analyze weather data using Hive statements
7.	Create an Internal Table, populate it with data and fetch the data
8.	Discuss about partitions and Buckets
9.	Write about alter and drop statements of hive
10.	Discuss hive tables with examples
11.	Explain the hive partitions with a detailed example
12.	Discuss about importing data in hive